DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
December 15, 1992	Final 20 Sep 91 - 19 Sep 92

4. TITLE AND SUBTITLE

Very High-Speed Report File System; Final Report

5. FUNDING NUMBERS

DAAL03-91-C-0049

6. AUTHOR(S)

Timothy J. Salo
John D. Cavanaugh
Joseph P. Thomas
Michael K. Spengler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Minnesota Supercomputer Center, Inc.
1200 Washington Ave. S.
Minneapolis, MN 55415

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 29396.1-MA

11. SUPPLEMENTARY NOTES

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Minnesota Supercomputer Center, Inc. (MSCI), under its Very High-Speed Remote File System (VHSRFS) contract with the Army Research Office, conducted research in:

- Technologies for integrating TCP/IP protocols with ATM networks;
- Technologies which will enable gigabit-per-second local area networks (LANs) and high-performance hosts to use gigabit-speed ATM wide-area networks;
- Methods by which high-performance, massively parallel processors (MPPs) can connect to high-speed wide-area networks; and
- Protocols and other technologies required to develop a shared, remote file system capable of transferring data to multiple remote supercomputers over gigabit wide-area networks.

The research results of the VHSRFS contract were intended to provide some of the foundation for research in the MAGIC Gigabit Testbed, which has been funded by DARPA.

93 2 18 035

14. SUBJECT TERMS

Gigabit Networks, Broadband Integrated Services Digital Networks (B-ISDN), Asynchronous Transfer Mode (ATM)

15. NUMBER OF PAGES

70

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Very High-Speed Remote File System

Final Report

Timothy J. Salo, Principal Investigator
John D. Cavanaugh
Joseph P. Thomas
Michael K. Spengler

December 15, 1992

U.S. Army Research Office
Contract Number DAAL03-91-C-0049

Minnesota Supercomputer Center, Inc.
1200 Washington Avenue South
Minneapolis, MN 55415
(612) 626-1888

Approved for Public Release;
Distribution Unlimited

Table of Contents

1.0 Foreword

The Minnesota Supercomputer Center, Inc's (MSCI's) Very High-Speed Remote File System (VHSRFS) project conducted research which evolved from the original focus on the design of a shared, remote file system which would be capable of transferring data to remote hosts at supercomputer speeds over gigabit wide-area networks. The VHSRFS was intended to be used by the terrain imagery application of the MAGIC Gigabit Testbed.

The initial objectives of the research included:

1. Survey and analyze available alternatives for remote file system protocols and recommend remote file system protocols for use by VHSRFS;

2. Develop a high-level design for a very high-speed, remote file system;

3. Analyze potential hardware and protocol technology for supercomputer channel extension, specifically, technology which extends HIPPI channels over ATM networks; and

4. Identify the software and other development activity required to enable additional supercomputer architectures, specifically, the Army High Performance Computing Research Center (AHPCRC) Connection Machine, (at the time a 32K processor CM-2), to communicate with the VHSRFS.

Several events occurred during the VHSRFS project which caused the focus of the research to be refined.

- Alternate technology was selected to provide mass storage for the MAGIC Gigabit Testbed.

- The researchers quickly determined that extending HIPPI connections across a wide-area network resulted in very poor performance.

- The AHPCRC took delivery of a CM-5 to replace its CM-2.

As a result of these events, the research evolved in several directions:

- The design of a remote file system received less attention than was originally expected.

- Researchers investigated methods of connecting a variety of high-bandwidth local network environments to gigabit wide-area networks, rather than limiting examination to extending HIPPI channels.

- Technology which would provide high-bandwidth network I/O for the CM-5 was pursued, rather than solutions for connecting the CM-2 to the VHSRFS.

The refined focus of this research helped ensure that the results met the objectives of the research proposal as well as address the needs of the MAGIC Gigabit Testbed.

2.0 Statement of Problem Studied

The Minnesota Supercomputer Center, Inc. (MSCI), under its Very High-Speed Remote File System (VHSRFS) contract with the Army Research Office, conducted research in:

- Technologies for integrating TCP/IP protocols with ATM networks;
- Technologies which will enable gigabit-per-second local area networks (LANs) and high-performance hosts to use gigabit-speed ATM wide-area networks;
- Methods by which high-performance, massively parallel processors (MPPs) can connect to high-speed wide-area networks; and
- Protocols and other technologies required to develop a shared, remote file system capable of transferring data to multiple remote supercomputers over gigabit wide-area networks.

The research results of the VHSRFS contract were intended to provide some of the foundation for research in the MAGIC Gigabit Testbed, which has been funded by DARPA.

MAGIC Gigabit Testbed

The MAGIC testbed will explore the unique challenges presented by applications which require intermittent gigabit/second data transfers. One example of this paradigm is the terrain mapping application which requires a supercomputer to process large volumes of data stored at one or more remote locations, and to distribute still frame and animated sequences interactively to remote users.

As described in the preliminary draft *MAGIC Gigabit Testbed Research Plan* dated 1/24/92, the testbed will include :

- A wide-area, asynchronous transfer mode (ATM) network connecting heterogeneous elements including supercomputer systems, gigabit LANs, graphics workstations, mass storage systems, multimedia facilities, and ATM switches;
- A prototype, DoD-oriented application that enables a user to drive through or fly over and see terrain;
- A scalable image server system that is distributed and of sufficient performance to support the visualization application;
- Data flow monitoring facilities at all levels of the image server system, the application, and the network to permit the experimentation with, and evaluation of, the interactions between the network and the distributed data and processing; and
- Multiple diverse supercomputer applications that require a wide-area gigabit internetwork.

The prototype terrain visualization application is expected to be hosted on the Army's CM-5 massively parallel processor located at the Army High Performance Computing Research Center (AHPCRC), based at the University of Minnesota. The US Army Battle Command Battle Laboratory (previously Future Battle Laboratory) will be the initial user of this application. In addition to the Minnesota Supercomputer Center, Inc., research participants of the MAGIC Gigabit Testbed include: EROS Data Center, Lawrence Berkeley Laboratory, SRI International and University of Kansas. Sprint will provide the wide-area fiber optic backbone.

3.0 Summary of Important Results

The result of the VHSRFS research are detailed in the attached technical reports.

The preliminary results of the VHSRFS research have helped define the protocols which are planned to be used in the MAGIC Gigabit Testbed: ATM, ATM Adaptation Layer 5 (AAL5), and TCP/IP. In a similar fashion, this work has refined our understanding of how high-performance hosts should connect to ATM wide-area networks, (e.g., HIPPI connections should terminate locally; the HIPPI/ATM gateway ought to be a layer three (e.g., IP) router, rather than a layer two device (ATM terminal adapter)).

Four technical reports are included as Appendices to this Final Report.

1. *Internetworking with ATM WANs*

   This report examines the issues which result from using ATM wide-area networks to interconnect high-performance LANs. The potential architectures for integrating LANs with ATM services are summarized and evaluated. Recommended architectures for connecting LANs and ATM directly applicable to the MAGIC Gigabit Testbed are included.

2. *Technologies for Gigabit Distributed File Systems*

   The requirements imposed on a distributed file system by wide-area gigabit networks are described in this report. The ability of existing distributed file system technologies, (e.g., NFS, AFS, and RFS) to meet these requirements are analyzed. The report recommends:

   - The most appropriate existing distributed file system software and protocol technology for use in a very high-speed distributed file system, and
   - Protocol enhancements for the recommended distributed file system protocol to support very high bandwidth, long propagation delay links.

3. *High-Performance Network I/O for a Massively Parallel Processor*

   The challenges of developing high-performance network I/O interfaces for massively parallel processors are examined. Recommendations, focused on the Thinking Machines Corporation CM-5, are included, although the analysis is applicable to a range of massively parallel processors.

4. *Design for a Very High-Speed Remote File System*

   A high-level design for the VHSRFS is summarized in this report. This design could guide the detail design and implementation of the VHSRFS.

4.0 Publications and Technical Reports

The research funded by this contract resulted in four technical reports. These technical reports are included in this Final Report as Appendices. The four technical reports are:

- Cavanaugh, John D. and Salo, Timothy J., *Internetworking with ATM WANs,* Minnesota Supercomputer Center, Inc., 1992.

- Spengler, Michael K. and Salo, Timothy J., *Technologies for Gigabit Distributed File Systems,* Minnesota Supercomputer Center, Inc., 1992.

- Thomas, Joseph P. and Salo, Timothy J., *High-Performance Network I/O for a Massively Parallel Processor,* Minnesota Supercomputer Center, Inc., 1992.

- Thomas, Joseph P., Cavanaugh, John D. and Salo, Timothy J., *Design of a Very High-Speed Remote File System,* Minnesota Supercomputer Center, Inc., 1992.

5.0 Participating Scientific Personnel and Advanced Degrees Earned

The research funded by this contract was undertaken by the Communications Group at the Minnesota Supercomputer Center, Inc. The following members of the MSCI Communications Group participated in this research:

Timothy J. Salo
John D. Cavanaugh
Michael K. Spengler
Joseph P. Thomas
Jeffery A. Wabik

No degrees were awarded to the participating scientific personnel during the research contract.

6.0 Reports of Inventions

No patentable inventions were developed under this contract. The principal result of the research is the technical reports which are included in this Final Report.

7.0 Bibliography

The attached technical reports include bibliographies. Refer to the technical reports for additional information.

Appendix A

Cavanaugh, John D. and Salo, Timothy J., *Internetworking with ATM WANs,* Minnesota Supercomputer Center, Inc., 1992.

This deliverable is referred to as "Analysis of Supercomputer Channel Extension Technology" in the VHSRFS proposal.

Internetworking with ATM WANs

John David Cavanaugh
Timothy J. Salo
Minnesota Supercomputer Center, Inc.*

December 14, 1992

Abstract

High-speed (155–622 Mb/s) wide-area networks based on the Broadband Integrated Services Digital Network's (B-ISDN) Asynchronous Transfer Mode (ATM) technology are currently being tested, and commercial service offerings are expected soon. This paper is an overview of ATM, ATM adaptation layers, and the Connectionless Broadband Data Service (CBDS). It discusses internetworking strategies and issues that arise when using ATM networks to carry the TCP/IP protocol suite.

1 Introduction

The Internet Protocol (IP) family (IP, TCP, UDP, Telnet, SMTP, FTP, etc.) has met with great success during the last decade. It is used on many different types of computers, from PCs to supercomputers. It is used in networks that fit in a single room, networks that serve a campus or corporation, and networks that span the globe. It uses a variety of media—including Ethernet[1], Token Ring, FDDI, X.25, high-speed computer channels, switched and leased telephone lines—ranging in speed from a few hundred bits per second to hundreds of megabits per second. It is used by a wide range of applications—file transfer, electronic mail, file sharing, and graphical user interfaces, to name just a few. It has proven effective in all these varied circumstances.

Work is now under way to adapt the IP protocol family to emerging network technologies that offer very-high-speed (up to the gigabit per second range) data transfer over long distances. Two of these new technologies are Asynchronous Transfer Mode (ATM) and Switched Multi-Megabit Data Service (SMDS). Both are public, switched data services and are expected to develop into networks of international scope, similar to the telephone network. ATM is connection-oriented; SMDS is connectionless. ATM is expected to be available at 155 and 622 Mb/s in the public network; SMDS is currently being tested at 1.5 and 45 Mb/s and is expected to reach 150 Mb/s.

These new technologies pose some challenges to the IP protocol family, mainly to the IP and TCP layers. A number of issues must be resolved before the IP protocol family can be used widely over these new networks. Some of these issues arise simply because the network media are new; some are due to the structure of the networks; some are caused by the speed of the networks. These issues (discussed in detail in sections 3 and 4) include the following:

- appropriate internetworking strategies
- format of encapsulation of IP datagrams
- network definition
- management of ATM connections
- TCP performance issues
- efficiency of datagram transmission

*The research described herein was sponsored by DARPA through the U. S. Army Research Office, Department of the Army, contract number DAAL03-91-C-0049. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by either DARPA or the U. S. Army.

[1]Ethernet is a registered trademark of Xerox Corporation.

Although many of these issues are common to several network technologies, this paper focuses on using the IP protocol family on ATM networks.

1.1 Organization of This Paper

Section 2 is an overview of ATM. Section 2.2 discusses ATM adaptation layers. Section 2.3 discusses the Connectionless Broadband Data Service (CBDS); readers who are already familiar with these topics may wish to skip these sections. Internetworking strategies for IP over ATM are discussed in section 3. Networking issues related to IP and ATM are discussed in section 4. Section 5 is a summary of the discussion and presents recommendations and conclusions.

Appendix A is a list of the acronyms used in this paper. Appendix B lists the CCITT recommendations for Broadband ISDN (B-ISDN).

2 An Overview of ATM

Asynchronous Transfer Mode[2] (ATM) is a technique for multiplexing fixed-length cells from a variety of sources to a variety of remote locations. ATM is capable of moving data at a wide range of speeds, but its usual application will be to carry data over optical fiber at very high speed (100–1000 Mb/s). It is capable of handling data from a variety of media (e.g. voice, video, and data) using a single interface and can multiplex a number of connections onto that interface.

ATM is a connection-oriented protocol. Connections are established between ATM service users, whether they are attached to the ATM network (e.g. a workstation with an ATM interface) or part of it (e.g. a call setup process running on an ATM switch). Connections can be switched, semi-permanent, or permanent. *Signalling procedures* are used to set up switched calls. These signalling procedures correspond to the dialing of a telephone. Semi-permanent and permanent connections are established through administrative procedures.

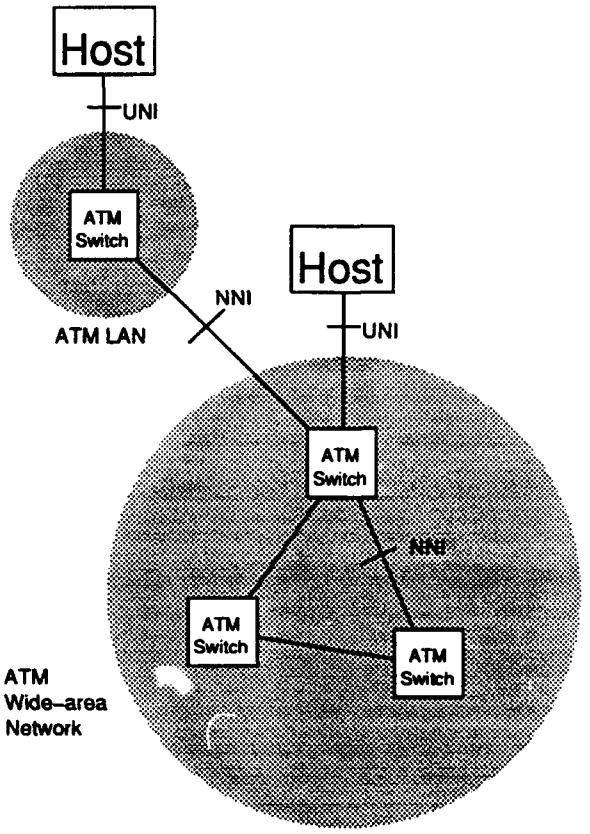

Figure 1: ATM Network Structure

The ATM service in wide-area networks is expected to run over a SONET carrier at SONET speeds (other media may be used in LANs). The speeds that are important to this discussion are OC-3c[3] (155.520 Mb/s), and OC-12c (622.080 Mb/s).

Each ATM connection operates at a certain *grade of service* (GOS). A particular GOS is characterized by such parameters as cell loss and cell delay variation. The GOS is negotiated between the user and the network when the connection is established. ATM operates on a

[2] Asynchronous Transfer Mode can be used to mean either the general technique of transferring data via fixed length cells or the particular version of that technique specified by the CCITT as the transport protocol for B-ISDN. In this paper, we use it in the latter sense.

[3] SONET's transmission rates are defined as multiples of 51.840 Mb/s OC-1 channels. An OC-3 channel is capable of carrying 3 OC-1 channels, or 3 x 51.840 Mb/s, 155.520 Mb/s. The designation 'c' indicates that the channels are concatenated; that is, they operate as a single channel rather than as n multiplexed independent channels. An OC-3 channel comprises three separate OC-1 channels, while an OC-3c channel is a single channel with three times the capacity of an OC-1 channel.

"best effort" basis; cells with errors, or that encounter congestion, are silently dropped. It is the responsibility of higher protocol layers to notice that cells are missing and, if required, to arrange for retransmission of the lost cells. A sequence of cells in an ATM connection will be received in the same order as it was transmitted; ATM guarantees that cells will not be misordered.

There are two types of connections: point-to-point and multipoint. Point-to-point connections are like ordinary two-party telephone calls; they connect two ATM service users. Multipoint connections are like conference calls; they connect more than two ATM service users. Multipoint connections can be used to provide a multicast facility.

Figure 1 shows a simplified ATM network. Network services are provided by ATM switches within the network. In an ATM LAN, the ATM switch will act as the network hub. Hosts are attached to the ATM network at the *user-network interface* (UNI). ATM switches are interconnected at the *network node interface* (NNI). Two distinct NNIs may be needed: one for the interconnection of ATM switches within the public network and a second for the attachment of ATM LAN switches to the public network.

2.1 ATM Cell

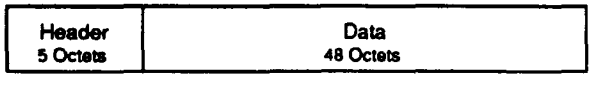

Figure 2: ATM Cell Structure

ATM transfers data in fixed-length units called *cells*. An ATM cell is 53 octets long—5 octets are header information and the remaining 48 are data. If the payload in a cell does not require the full 48 octets, padding is used (by the layer above ATM) to fill the cell. The 48-octet size represents a compromise between the demand of voice traffic for quick access to the network and the demand of data traffic for large data units. Figure 2 shows the layout of the ATM cell; Figure 3 shows the layout of the ATM header.

The fields in the header have the following uses:

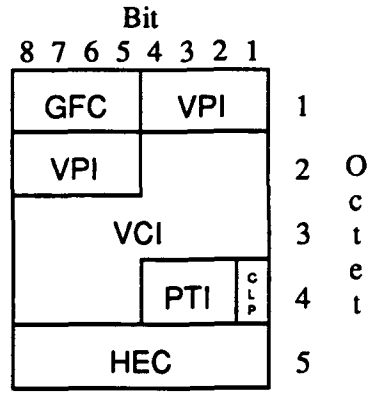

Figure 3: ATM Header

GFC Generic Flow Control. 4 bits. Used by the flow control mechanism at the UNI. Mechanisms are yet to be determined.

VPI Virtual Path Identifier. 8 bits. Used for directing cells within the ATM network (see discussion below).

VCI Virtual Channel Identifier. 16 bits. Used for directing cells within the ATM network (see discussion below).

PTI Payload Type Indicator. 3 bits. Identifies the type of data being carried by the cell. See Table 1 for values.

CLP Cell Loss Priority. 1 bit. If this bit is set (equal to 1), the cell has low priority and is subject to being discarded when the network is under stress. If it is not set (equal to 0) the cell has higher priority and is less likely to be discarded.

HEC Header Error Correction. 8 bits. Generated and inserted by the physical layer. Serves as a checksum for the first 4 octets of the ATM header. It can correct single-bit errors and detect some multiple-bit errors.

ATM cells flow along entities known as *virtual channels* (VCs). A VC is identified by its *virtual channel identifier* (VCI). All cells in a given VC follow the same route across the network and are delivered in the order they were transmitted. A VC between two users can carry data and signalling information. A VC between a user and the ATM network can be used for signalling or for administrative purposes. Three VCs—numbers 0, 1, and 2—are reserved for

special purposes. 0 is for unassigned cells; 1 indicates the meta-signalling VC; 2 indicates the general broadcast signalling VC.

VCs are transported within *virtual paths* (VPs). A VP is identified by its *virtual path identifier* (VPI). VPs are used for aggregating VCs together or for providing an unstructured data pipe. Depending on the requirements of the user and the network, it is possible that not all bits of the VPI and VCI will be significant. Bits that are not significant are set to zero.

The VPI and VCI, taken together, form a 24-bit *protocol connection identifier* (PCI). The PCI identifies a particular call or connection and is used for routing cells across the network and demultiplexing cells at the destination. Cells with PCI equal to zero are *unassigned* or unused. They carry no user data.

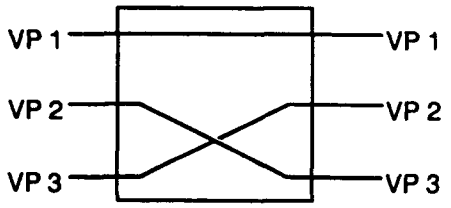

Figure 4: VP Switch

VPs and VCs are subject to switching within the ATM network. A VP switch can redirect a VP, perhaps reassigning the VPI, but keeps the VCs within the VP intact (see Figure 4).

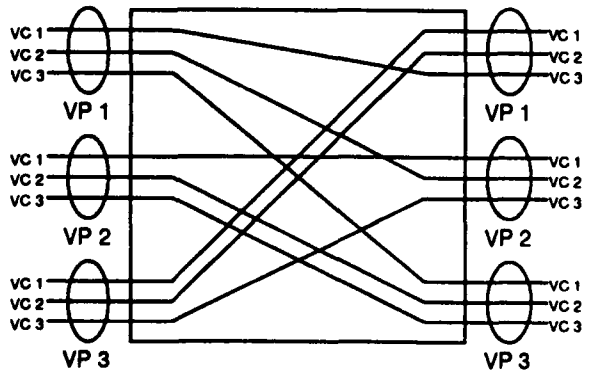

Figure 5: VC Switch

A VC switch must terminate VPs and can switch the VCs within a VP independently of each other (including reassigning their VCIs; see Figure 5).

All this switching means that the VPI and VCI at different ends of a connection may not be the same. That is, each end point of a connection (and there may be more than two for a multipoint connection) may refer to the same connection with a different VPI or VCI.

The type of information carried in each cell is denoted by the PTI field. The meanings of the different values of this field are shown in Table 1.

000	User data, congestion not experienced, Segmentation Data Unit (SDU) type 0
001	User data, congestion not experienced, SDU type 1
010	User data, congestion experienced, SDU type 0
011	User data, congestion experienced, SDU type 1
100	Segment Operation and Maintenance (OAM) F5 flow-related cell
101	End-to-end OAM F5 flow-related cell
110	Resource management cell
111	Reserved for future use

Table 1: Payload Type Indicator Encoding

Note that in user data cells (PTI = 0XX), the value of the third bit indicates the Service Data Unit (SDU) type. This is used by AAL 5 to indicate the end of a data unit (see section 2.2.2).

The use of cells with PTI types 100, 101, 110, and 111 is for further study. Cells with PTI types 100 and 101 are Operation and Maintenance (OAM) F5 cells; these are cells that carry administrative messages related to the operation and maintenance of the virtual circuit that carries them.

2.2 ATM Adaptation Layers

In order to carry data units longer than 48 octets in ATM cells, an adaptation layer is needed. The *ATM adaptation layer* (AAL) provides for segmentation and reassembly of higher-layer data units and for detection of errors in transmission.

Since the ATM layer simply carries cells without concern for their contents, a number of different AALs can be used across a single ATM interface. The end points of each connection must agree on which AAL they will use, but the network need not be concerned with this.

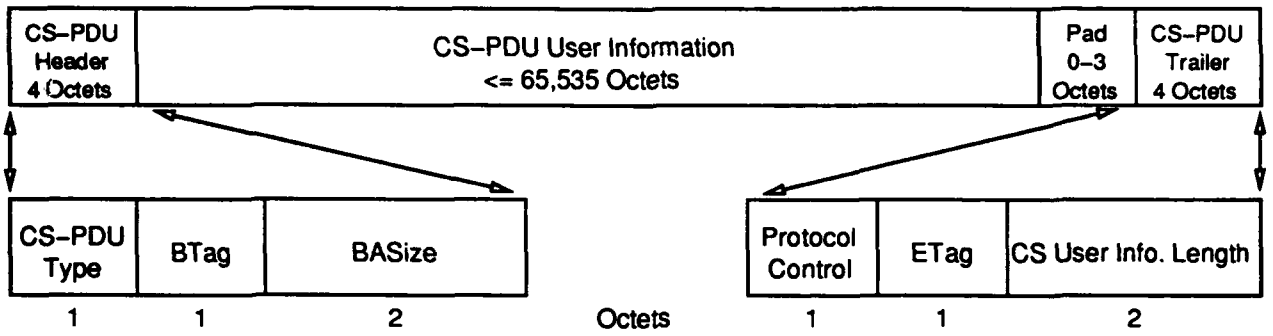

Figure 6: AAL 3/4 Convergence Sublayer (CS) PDU Structure

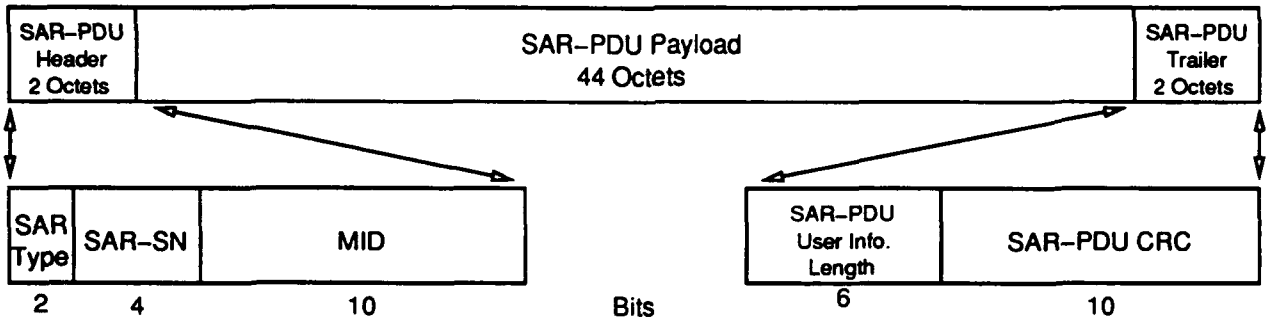

Figure 7: AAL 3/4 Segmentation and Reassembly (SAR) PDU Structure

Five AALs have been accepted for consideration by the CCITT. They are referred to as AAL 1–5. AAL 1 is meant for constant-bit-rate services (e.g. voice). AAL 2 is meant for variable-bit-rate services with a required timing relationship between source and destination (e.g. audio or video). AAL 3 was originally meant for connection-oriented variable-bit-rate services without a required timing relationship; it has now been merged with AAL 4. AAL 3/4 and 5 are meant for connectionless services (e.g. connectionless data). Only AALs 3/4 and 5 are of interest for IP networking.

2.2.1 AAL 3/4

AAL 3/4, the *variable bit rate* (VBR) adaptation layer, defined in CCITT recommendation I.363, is defined for services (e.g. data) that require "bursty" bandwidth. AAL 3/4 actually comprises two sublayers, the *convergence sublayer* (CS) and the *segmentation and reassembly* (SAR) sublayer. The CS protocol data unit (PDU) structure is shown in Figure 6; the SAR PDU structure is shown in Figure 7.

The pad field in the CS-PDU ensures that the length of the PDU is divisible by 4. The value of the pad field is binary zeroes.

The fields in the CS header and trailer have the following uses:

CS-PDU Type 1 octet. Use under study. When the upper layer is connectionless, this field is set to zero.

BTag 1 octet. Use under study. Meant to be used in conjunction with ETag.

BASize 2 octets. Contains the length in octets of the CS-PDU user information.

Protocol Control 1 octet. Use under study. When the upper layer is connectionless, this field is set to zero.

ETag 1 octet. Use under study. Meant to be used in conjunction with BTag.

CS-PDU User Information Length 2 octets. The length of the user information in the CS-PDU (not including pad).

As indicated by the number of fields described as "Use under study," the formulation of the AAL 3/4 CS is still incomplete.

The fields in the SAR header and trailer have the following uses:

SAR Type 2 bits. Differentiates the parts of a segmented message. Values are:

01	beginning of message (BOM)
00	continuation of message (COM)
10	end of message (EOM)
11	single-segment message (SSM)

SAR-SN 4 bits. Indicates the position of the segment within the message[4]

MID 10 bits. Identifies the message. All segments of a given message will have the same MID value.

SAR-PDU User Information Length 6 bits. Gives the length of the user information in the SAR-PDU.

SAR-PDU CRC 10 bits. A CRC that covers the entire SAR-PDU.

When a network node has a user datagram to transmit, it first converts it to a CS-PDU by adding the CS-PDU header, pad (if necessary), and trailer. It then splits the CS-PDU into 44-octet chunks and converts each chunk to a SAR-PDU by adding the SAR header and trailer, and transmits them, one SAR-PDU per ATM cell. If necessary, the last chunk of the CS-PDU is padded to fill out the last SAR-PDU.

When SAR-PDUs are received, they are reassembled into CS-PDUs, with the receiver using the SAR Type, SAR-SN, and MID fields to ensure that the chunks return to their proper places. The receiver verifies that the reassembled CS-PDU is intact using the fields in the CS-PDU header and trailer (e.g. the length of the CS-PDU as reassembled by the SAR sublayer is compared to the length value in the CS-PDU trailer).

[4] The SAR-SN indicates the sequential position, modulo 16, of the segment in the message.

2.2.2 AAL 5

AAL 5, the Simple and Efficient Adaptation Layer (SEAL) [1], attempts to reduce the complexity and overhead of AAL 3/4. It eliminates most of the protocol overhead of AAL 3/4. Like AAL 3/4, AAL 5 comprises a convergence sublayer and a SAR sublayer, although the AAL 5 SAR sublayer is essentially null. The structure of the AAL 5 CS PDU is shown in Figure 8.

The pad field in the CS-PDU ensures that the total length of the CS-PDU (including pad and trailer) is divisible by 48 and that the CS-PDU trailer resides in the last eight octets of the last 48-octet chunk. The value of the pad field is binary zeroes.

The fields in the AAL 5 CS-PDU trailer have the following uses:

Control 16 bits. Use under study.

Length 16 bits. The length of the user information in the CS-PDU (not including pad).

CRC 32 bits. A 32-bit CRC (CRC-32; the same as that used in FDDI and Fibre Channel) covering the data and pad.

When a network node has a user datagram to transmit, it first converts it to a CS-PDU by adding the pad (if necessary) and trailer. Then it breaks the CS-PDU into 48-octet SAR-PDUs and transmits each in an ATM cell on the same virtual channel. The last SAR-PDU is marked so that the receiver can recognize it. Since there is no AAL 5 SAR header, an end-of-frame indication in the ATM cell header is required. The proposed end-of-frame indication is an SDU type of 1 (binary value '0X1') in the Payload Type Indicator (PTI) field.

The receiver simply concatenates cells as they are received, watching for the end-of-frame indication. When it is seen, the receiver checks the length and CRC and passes the PDU up to the next higher layer. The higher layer is responsible for ignoring PDUs with CRC errors. Some applications may discard PDUs with errors; others (e.g. voice or video, which can

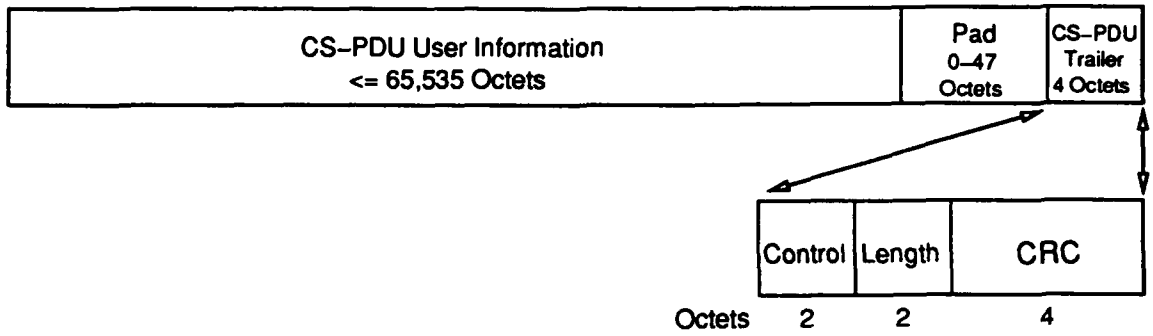

Figure 8: AAL 5 PDU Structure

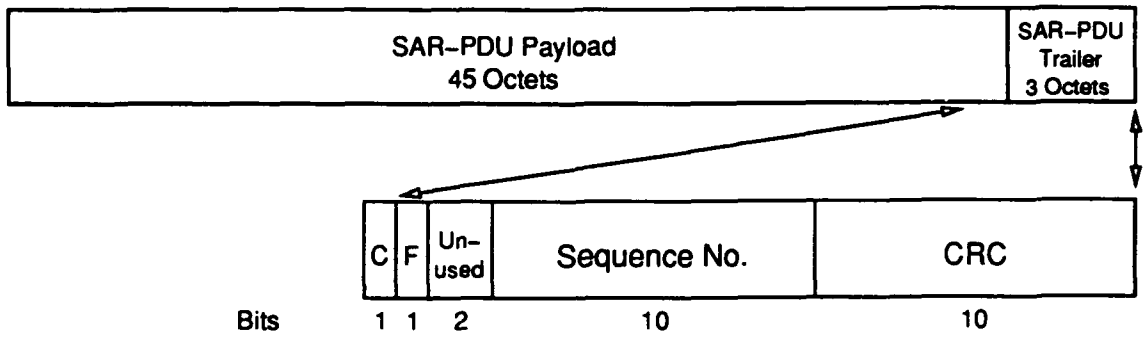

Figure 9: BBN SAR PDU Structure

tolerate some errors) may choose to use them, or to do error correction.

2.2.3 Other AALs

In addition to AAL 1–5, two other AALs have been defined. The first, BLINKBLT [13], provides the complete OSI data link layer service across an ATM network. Unlike the other AALs, it contains provisions for recovery of lost cells.

BLINKBLT comprises two sublayers, the cell and frame sublayers, which correspond roughly to the CS and SAR of AAL 3/4. Eight PDU types are defined; two are for the transport of user data, the others are control messages.

BLINKBLT is unique in that, unlike the other AALs, it is connection-oriented. Connections are established by a three-way handshake and closed by a similar mechanism. The connection is considered to consist of two one-way circuits, each with its own sequence number space. Within each half-circuit, the receiver controls the data rate by granting credits to the transmitter. Each correctly-received cell is explicitly acknowledged; cells that are lost or received with errors are retransmitted. Cells are protected by a 10-bit CRC[5] and frames are further protected by a 32-bit CRC.

The second alternative AAL, the BBN SAR protocol [11], is an experimental segmentation-and-reassembly protocol that was defined at Bolt Beranek and Newman, Inc. (BBN). The BBN SAR was designed to replace the existing profusion of AALs with a single protocol. It was designed to work with an undefined (and possibly empty) convergence sublayer. These are the four main features of the BBN SAR:

- detection of lost or misordered cells via a sequence number
- error correction via a 10-bit CRC
- demarcation of higher-layer data units via a frame bit

[5] The same CRC is used by AAL 3/4.

- provision for insertion of control cells into the data stream

Control information for the BBN SAR is held in a three-octet trailer inserted into each ATM cell. Figure 9 shows the layout of the trailer and its position in the cell.

The fields in the BBN SAR trailer have the following uses:

C 1 bit. The control-cell bit. Set to 1 if the cell is for control; set to 0 if the cell carries user data.

F 1 bit. The frame bit. Set to 1 if this is the last cell of a frame; set to 0 otherwise.

Unused 2 bits. Currently unused.

Sequence No. 10 bits. A counter that is incremented (modulo 1024) by 1 for each cell sent over an ATM connection.

CRC 10 bits. A CRC[6] for detection and correction of errors[7].

Transmission and reception of SAR-PDUs is similar to that of AAL 3/4 and 5. The transmitter breaks the higher-layer data unit into chunks, appends the SAR trailer, and transmits them, while the receiver reassembles the chunks into the original data unit. The main difference is the introduction of a window based on the sequence number.

The trailing edge of the window is the next sequence number the receiver expects. The leading edge is set by the application to the trailing edge plus the number of cells the receiver is prepared to buffer. During normal data transfer, the window moves forward by one each time a cell is received. When a cell is lost or misordered, the window freezes. The receiver accepts and stores cells whose sequence numbers fall within the window until the window fills, a time-out occurs, or the missing cell is received. In either of the first two cases, the receiver reports a lost cell to the next higher layer and repositions the window past the lost cell. In the third case, the receiver passes all the received cells to the higher layer and repositions the window.

Both BLINKBLT and the BBN SAR were submitted to ANSI for consideration. However, ANSI did not choose either as the basis for a standard, so neither was submitted to CCITT.

[6] Again, the same CRC as is used by AAL 3/4.

[7] In AAL 3/4, the correction of errors is optional, in the BBN SAR, it is required.

2.3 Connectionless Broadband Data Service

In order to provide a connectionless data service over an ATM network, the Connectionless Broadband Data Service (CBDS) has been defined in draft recommendation I.364. An example ATM internetwork supporting CBDS is shown in Figure 10.

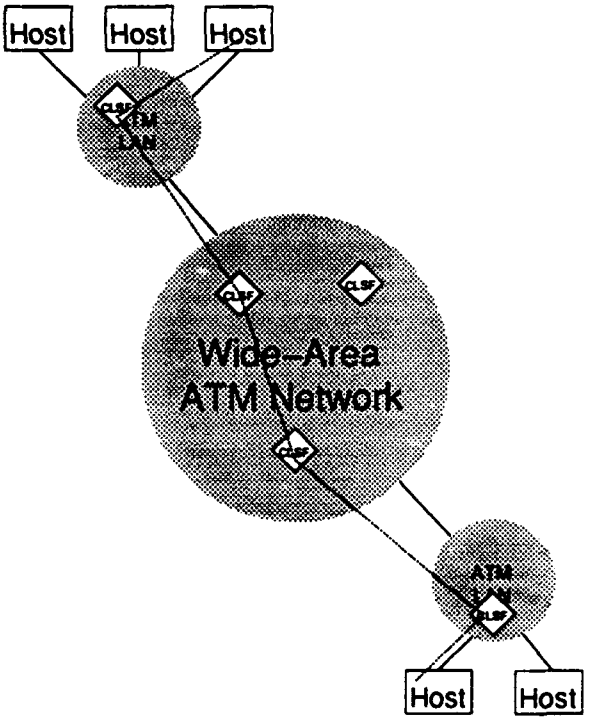

Figure 10: CBDS Operation

As shown in Figure 10, a number of *connectionless service functions* (CLSFs) exist throughout the network. The CLSFs act as cell-based routers, forwarding connectionless data units (CL-PDUs) across the network.

To send a CL-PDU, an ATM station does not attempt to make an ATM connection to the destination. Instead, it sends the PDU to a

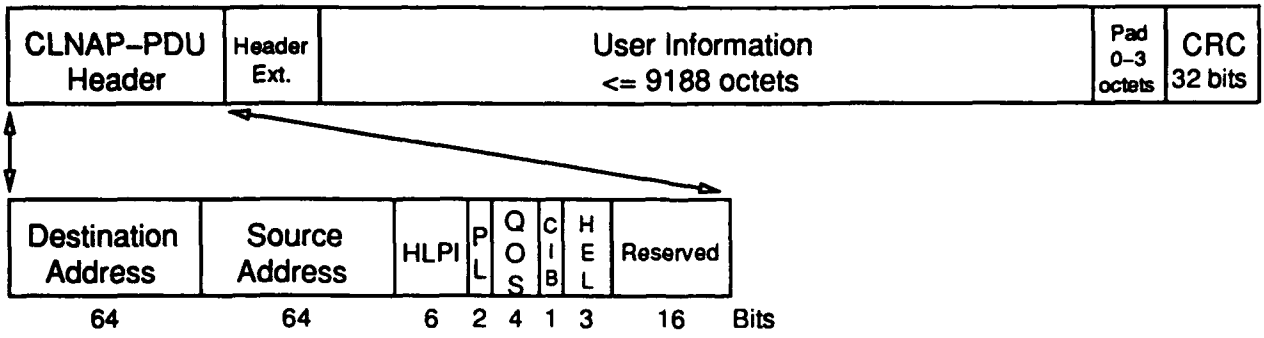

Figure 11: CLNAP PDU Structure

CLSF, opening a connection to it if necessary (it is probably preferable that the station already have an open (semi-) permanent connection to the CLSF). The receiving CLSF relays the CL-PDU to another CLSF closer to the destination, and so the CL-PDU "hops" its way across the network until it reaches a CLSF that can deliver it to the destination. The CLSFs do not reassemble the CL-PDU at each hop, but "stream" the ATM cells comprising the CL-PDU. The CL-PDU is only reassembled at its destination.

The protocol used by the CLSFs is the Connectionless Network Access Protocol (CLNAP). The structure of the CLNAP PDU is shown in Figure 11.

The fields in the CLNAP PDU have the following uses:

Destination Address 64 bits. The E.164 address of the destination of the data.

Source Address 64 bits. The E.164 address of the source of the data.

HLPI Higher Layer Protocol Identifier. 6 bits. Indicates the entity that is to receive the data at the destination. Carried transparently across the network.

PL Pad Length. 2 bits. Indicates the number of pad octets in the PDU.

QOS Quality of Service. 4 bits. Indicates the quality of service requested by the source. For further study.

CIB CRC Indicator Bit. 1 bit. Set to 1 to indicate the presence of the optional CRC. For further study.

HEL Header Extension Length. 3 bits. Indicates the number of 32-bit words in the header extension field.

Reserved 16 bits. Set to zero.

Header Extension Variable length. For further study.

User Information Variable length. The data being transported on behalf of the CLNAP user.

Pad Padding. 0–3 octets. Sufficient padding to make the length of the user information plus pad divisible by 4.

CRC Cyclic Redundancy Code. 32 bits. For further study.

The format of the source and destination addresses is as specified by E.164. E.164 is a CCITT recommendation that specifies the worldwide ISDN numbering plan.

The draft recommendation specifies that the CLNAP-PDUs would be encapsulated by AAL 3/4 to be transmitted across an ATM network.

2.4 Unresolved ATM Issues

ATM is a young and evolving standard. Significant parts are still not in place. Some of these have effects only within the ATM network:

others are more visible. This section discusses a couple of the most significant uncertainties visible to upper layers.

The most obvious unresolved issue for ATM is the question of AALs. No AAL standard is complete—the AALs that have been defined are still subject to change—and no AAL has been approved for use with IP. This is an important issue because network devices that wish to interoperate have to use the same AAL (and, of course, agree on what that means).

The setting of standards for addressing and signalling within ATM networks is also important. Until addressing and signalling are defined, ATM cannot be widely used, as equipment from different vendors will probably not interoperate. One effect of this is that the interoperation of ATM LANs with the public ATM network cannot be defined. We expect that ATM LANs will eventually appear to be a part of a large, seamless ATM network (see also section 3.1), but in the absence of standards for signalling procedures, they can only be connected to the public ATM network through network-level devices (routers).

3 Internetworking Strategies

The purpose of computer networks is to allow hosts to intercommunicate. To this end, the hosts have to be connected to the network. In the usual model, a host is connected to some sort of LAN, with LANs being connected into an *internetwork* by a wide-area network. An example of such an internetwork is shown in Figure 12.

We assume that this internetwork model applies (i.e. that hosts will continue to be connected to LANs, and that the LANs will be interconnected by wide-area networks) even with the advent of high-speed ATM networks. These are our reasons:

- A connection to the public ATM network is expected to be expensive; the internetwork model allows the expense to be shared by a large number of users.

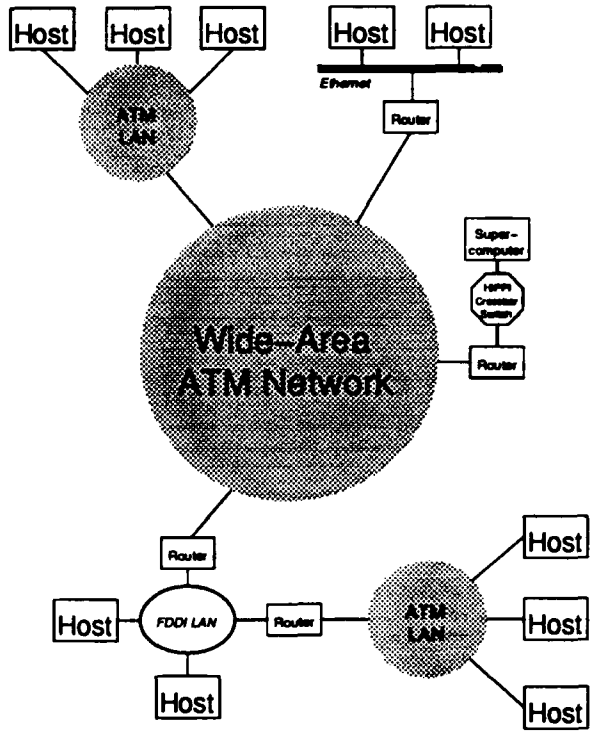

Figure 12: ATM-based Internetwork

- It is preferable for traffic between machines at a single site to cross a LAN rather than the public ATM network.
- There may be hosts for which an ATM interface is not available.
- There are many existing networks that use non-ATM media; these networks will still be in use after ATM networks are built.

3.1 ATM LAN–WAN Connection

As the public carriers are experimenting with ATM and installing ATM switches, vendors are producing small ATM switches for use in LANs. These devices are different from other LAN technologies (e.g. Ethernet, FDDI, HIPPI) in that they can potentially interoperate directly with ATM switches in the public network. An interesting question is whether, once standards for addressing, signalling, billing, etc. are in place, local ATM switches will be developed so that they can interoperate with the public network.

If they are, we can expect that ATM LANs will

be integrated more or less seamlessly into the public ATM network, so that a host on an ATM LAN will be able to open a connection directly to another host on a different ATM LAN without concerning itself with the public network that interconnects the LANs. This arrangement is similar to the use of today's PBX. Its advantage is that a host on an ATM LAN could make direct ATM connections to remote hosts across the public ATM network but still communicate with local hosts without having to cross the public network.

Its disadvantages are the complexity of the LAN switch and the assignment of addresses. The NNI of the ATM LAN switch has to conform to the standards of the public ATM network if integration is to be complete. In contrast, a non-integrated ATM LAN switch does not need a NNI at all, but would be connected to the outside world through a router attached to a UNI on the switch. Also, if ATM LAN nodes are directly addressable from the public ATM network, the public network must be cognizant of the ATM-level addresses of those nodes. This implies a high degree of cooperation between the LAN and the public network, particularly in the assignment of addresses. In one scenario, ATM addresses (assigned in accordance with CCITT recommendation E.164) are administered by the operators of the public network and anyone wishing to connect an ATM LAN to the public ATM network has to request addresses. In another [21], E.164 addresses are used in the public network, but layer 3 addresses (e.g. IP addresses) are used in LANs, with a switch at the boundary being responsible for the required translations.

If ATM switches are not fully integrated, they will, like other types of LANs, need some intermediate device (bridge or router) between them and the public ATM network. This case is discussed below.

3.2 Bridging/Routing

It is tempting to transport high-speed data from existing channels (e.g. HIPPI) by extending the channel across the ATM network. The signals and data would simply be packed into ATM cells, shipped across the network, and turned back into HIPPI signals and data by an identical device on the other side. This approach is simple and direct. However, it has drawbacks that make it impractical:

- Performance limitations because of end-to-end propagation of signals.
- The need for compatible equipment on both ends of the connection.
- Potential for loss of HIPPI signals (e.g. READY) as they cross the network.

We believe that connecting to the public ATM network through a router, rather than a bridge, has the advantage of providing isolation and increasing interoperability.

Isolation: A router provides isolation from the data link layer of the public ATM network and thereby from any low-level misbehavior on the part of the public network or an attached host or remote network. Also, the low-level (data link or physical layer) addresses of network nodes behind the router are independent of those within the public ATM network, so LAN administrators could select addresses without worrying about the public network's numbering policy. A router can also perform network-level filtering of packets, providing an important security checkpoint.

Interoperability: Given that standards for IP exist for the media involved, a router improves interoperability. Equipment from different vendors can interoperate, and there is no requirement for LANs at the ends of a connection to be of the same type. For instance, a packet could originate at a host, cross a HIPPI LAN to a router, be sent across the public ATM network to a second router (built by a different manufacturer), and cross an ATM LAN to its destination.

4 ATM/IP Networking Issues

The IP protocol suite is used on a very large number of computers of different types. As local- and wide-area ATM networks become available,

they will be expected to carry IP traffic. This section examines some of the issues involved in using ATM networks, especially wide-area ATM networks, for IP.

4.1 Format of IP Encapsulation

If IP datagrams are to be transported within ATM cells, there must be agreement on the format of the encapsulation. There is currently no standard for sending IP over ATM. A couple of proposals have been put forward [9, 15], and the Internet Engineering Task Force (IETF) has taken up the issue, but a definitive answer is probably some time away. The basic issues are the choice of AAL and a method for identifying the higher-layer protocol.

The choices for adaptation layer are AAL 3/4 and AAL 5. The first proposals specified AAL 3/4, but AAL 5 seems to be the current favorite because of its simplicity and the data integrity offered by its 32-bit CRC.

Three proposals for identifying the higher-layer protocol have been made. The first uses a different virtual circuit for each protocol. Its advantage is that the format of the transmitted data is very simple; its disadvantage is the static nature of the assignment. The second proposal includes a *network level protocol ID* (NLPID) at the start of each packet: protocols without an assigned NLPID[8] are identified by an IEEE 802.1a Subnetwork Access Protocol (SNAP) header. The third proposal identifies the protocol by using an IEEE 802.2 *logical link control* (LLC) header. Again, provision is made for use of a SNAP header for protocols that cannot be identified by the LLC header.

4.2 IP Routing

The first part of an IP address specifies a *network number*. In general, a network number corresponds to a physical network, although the concept of *subnetting* allows a single network to be split into a number of physically distinct *subnets*. Physically distinct networks are interconnected by *routers*, which forward IP datagrams between (sub-)networks. IP assumes that if two hosts have IP addresses with different (sub-)network parts, it is not possible to transmit data directly between them.

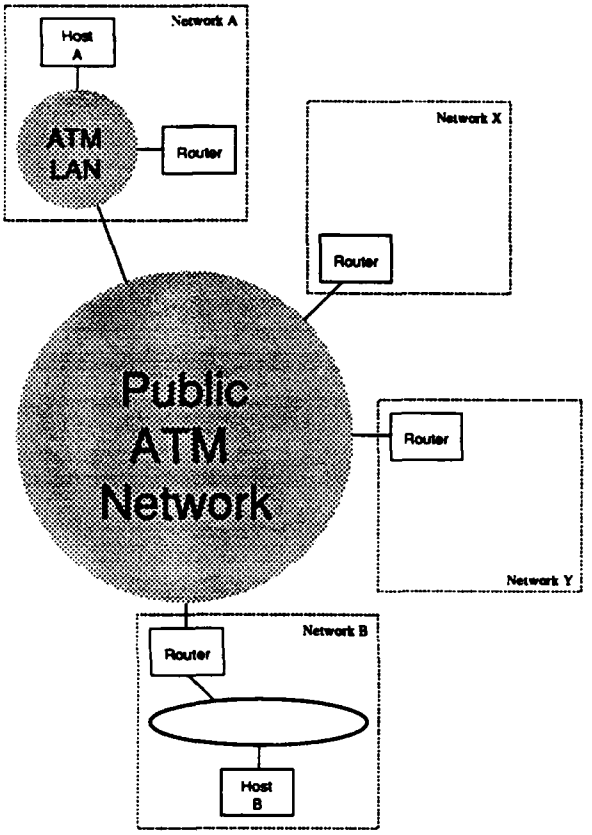

Figure 13: IP Routing Example

However, with a large number of networks all being connected to one public network, this may not be true. If both hosts have interfaces to the public network, they can, indeed, exchange data directly. As shown in Figure 13, Host A can, in fact, make an ATM connection directly to the router in network B, which means that Host B is effectively one hop away from Host A. If Host A is not aware of this, however, it will send its datagram to the router within its own network for forwarding. This router may not be aware of the router in network B and send the datagram to another router instead. In this way, the datagram may make several transits of the public ATM network on its way from Host A to Host B, when one would have been sufficient.

This problem is not unique to B-ISDN ATM networks; SMDS faces the same difficulties. In

[8] NLPID values are administered by ISO and CCITT. Values are defined in [17].

fact, a technique called *shortcut routing* [26] has been proposed for use in SMDS networks and may prove useful in ATM networks.

4.3 IP Address Resolution

Each host (or router) attached to an ATM network has one or more IP addresses. Each network interface also has an ATM address. A host that has a datagram to send knows the IP address of the destination; before it can send the datagram, it must learn the corresponding ATM address to locate (or create) a virtual channel that it can use to send the datagram.

On most LANs, a protocol known as the *address resolution protocol* (ARP) is used for this purpose. An ARP request is broadcast to all hosts on the LAN and the host that owns the address returns a response. Broadcasting is impractical (if not impossible) on a network as large as the public ATM network, so some other method of address resolution will have to be used.

Several different techniques be used. First, each network node keeps a table giving the correspondence between IP and ATM addresses for all the nodes on the network. For a network of any size, this is a huge administrative headache and clearly impractical. Second, a multipoint ATM connection is made to all nodes on a given network. This is practical if the network is small or if a group address is defined for all the nodes belonging to a particular IP network. The ARP request (and reply) is transmitted on the resulting multipoint VC. Third, a technique called *directed ARP* [12] uses the normal ARP packet formats but allows routers to forward an ARP request to another machine when there is reason to believe that it is capable of answering the request. Directed ARP allows a network node to test whether a target address is on the local physical network by simply sending an ARP request.

This difficulty with address resolution is not unique to ATM networks; SMDS faces the same problem.

4.4 ATM Connection Management

When switched connections are available, network hosts may want to make temporary connections to each other. How should these connections be managed?

One obvious solution is to "nail up" connections between hosts. These permanent connections would, like leased lines, always be available. This solution will probably be used between some hosts that have heavy traffic between them. For example, a permanent multipoint connection can be used to connect all the routers for a particular IP network that have interfaces to the public ATM network; this connection serves as a multicast server between the routers. However, it is not a general solution because it is not practical to nail up connections between a given host and all the other hosts it with which it might communicate.

Another solution is to establish connections on an "as needed" basis. When a host has a datagram to send, it checks to see whether it currently has an open connection to the destination. If it does, it uses the connection to send the datagram. If not, it stores the datagram, opens a connection, and then sends the datagram. If it cannot open a connection, it discards the datagram. If a connection is idle for a given time, it closes it. This scheme is already in use over X.25 networks. The drawback is that it increases the time a host takes to send a packet.

Ultimately, there will be a need for both types of connections. Hosts that constantly exchange a lot of data (e.g. routers) require permanent connections, while switched connections are appropriate for use between hosts that rarely communicate (e.g. only require connections for email transfer). The tariffs imposed for use of the public ATM network will also have an effect. High prices for permanent connections will encourage the use of switched connections and vice versa.

4.5 TCP Considerations

There is potential for disharmonious interactions between ATM and higher-layer protocols.

Cell loss: ATM discards cells when it encounters congestion. When an upper layer (e.g. TCP) notices that data was lost, it retransmits any unacknowledged data. The retransmitted data may be considerably more than was discarded. This has the potential for increasing the congestion, with obvious unfortunate consequences. Techniques such as slow-start and exponential retransmit timer backoff [18] should alleviate the damage, but research in live networks is needed.

TCP window management: Van Jacobson [20] states that standard TCP window algorithms run into trouble when the bandwidth × delay product of a link exceeds 10^5. On 2,500 miles of fiber, the round-trip delay is approximately 30 ms, giving a bandwidth × delay product of about 1.6×10 for an OC-12c circuit (for a 1.5 Mb/s satellite circuit, the bandwidth × delay product is about 7.7×10^5). Clearly, versions of TCP with high-speed modifications (extended windows, etc.) are needed to make effective use of high-speed ATM networks. Research is needed to determine whether current enhancements are adequate for the task.

Sequence number wrap: Under TCP, each octet of data is numbered. This *sequence number* is used by TCP's acknowledgement mechanism. On an OC-12c circuit transmitting at full speed, it would only take about 27 seconds for the 32-bit sequence number to wrap around and start reusing old values. A mechanism has been proposed to prevent this from causing problems [19].

Efficiency: Consider a null TCP data segment (e.g. an empty ACK). When this segment is transmitted over the ATM network, it consists of a (typically) 20-octet TCP header and a (typically) 20-octet IP header, so its length will be 40 octets. The AAL convergence sublayer occupies 8 octets (for either AAL 3/4 or AAL 5). If AAL 3/4 is used, the SAR takes 4 octets per cell, so even a null TCP segment doesn't fit in a single cell. If AAL 5 with no further encapsulation is used, a null TCP segment barely fits. If some further encapsulation (e.g. NLPID/SNAP) is used, no TCP segment fits in a single cell.

According to [8], about 40% of TCP wide-area traffic consists of empty ACKs; another 30% consists of TCP segments with 1–10 octets of data. These small segments end up being transmitted in two ATM cells, the second of which consists mainly of padding. This drives down the efficiency of the network. Various types of compression can improve efficiency; it remains to be seen whether this is really needed.

4.6 Performance

The speed that ATM networks are expected to attain makes performance a serious issue. In addition to the performance effects of TCP windows and transmission efficiency (discussed above), there is the issue of the load that the network can place on attached devices. At OC-12c speed (622.080 Mb/s, with 599.040 Mb/s available to the ATM layer), about 1.4 million cells per second are delivered to the UNI. Table 2 shows the *maximum transmission units* (MTUs) of various network media, along with the minimum and maximum IP packet sizes. For each packet size, the number of cells per packet and packets per second in the full OC-12c bandwidth are shown. Note that both AAL 3/4 and AAL 5 allow packets of the maximum size (65,535 octets).

Description	MTU	cells/pkt	pkt/s
Minimum	20	1	1,412,830
Generic	576	13	108,679
Ethernet	1,500	32	44,151
FDDI	4,352	91	15,526
HIPPI	65,280	1,361	1,038
Maximum	65,535	1,366	1,034
This table assumes AAL 5 encapsulation			

Table 2: Packet Sizes and Cell Rates

A common technique for improving communications performance is increasing the packet size. This reduces the number of interrupts that need to be handled and decreases the proportion of bandwidth used by overhead (packet headers, etc.). Using the maximum datagram size permitted by IP, a router only needs to handle a little over a thousand packets

per second. Router vendors can deliver products today that perform at this level, assuming that the ATM and AAL layers are implemented in hardware so there is not a per-cell demand on the router's CPU. It is, however, important to remember that traffic studies [7, 8] indicate that minimum-size packets make up a significant fraction of wide-area traffic.

4.7 Scalability

Today's experiments with ATM are being conducted on very small networks. As ATM is deployed throughout the network, these "islands" of ATM will grow and join, eventually evolving into a world-wide network. It is critical that the strategies, algorithms, and standards that are tested remain viable as they are scaled up to run over a global network.

5 Conclusion

ATM is an emerging technology that promises to figure heavily in both local- and wide-area networks. It is based on the transfer of fixed-length *cells* and allows for multiplexing through the use of *virtual circuits*. ATM is connection-oriented, but a proposed associated technology known as CBDS is connectionless. There are a number of unresolved issues associated with ATM, but quite a bit of work is being invested in it.

ATM networks need to interoperate, both with each other (e.g. LAN-WAN interoperation) and with networks based on other technologies (e.g. HIPPI, FDDI, Ethernet). While data-link-layer interoperation may be appropriate between networks based on ATM, network layer routing is more appropriate for interoperation between ATM and non-ATM networks. However, there are currently no commercially-available routers with adequate performance to handle routing between high-speed networks (e.g. a HIPPI LAN and a 622 Mb/s ATM WAN).

The introduction of a very-high-speed public wide-area network raises a number of issues for the IP family of protocols, including:

- internetworking strategies
- IP encapsulation
- optimization of routing
- IP address resolution
- connection management
- TCP considerations
  - cell loss
  - window management
  - sequence number wrap
  - efficiency
- performance
- scalability

Network research is needed to resolve these issues and to answer outstanding questions about running TCP/IP over ATM.

For the near term, these are our recommendations for building an IP internet which makes use of ATM:

Use AAL 5 as the ATM adaptation layer. It is designed to be efficient at transporting data, it provides excellent error detection, and it is the most likely AAL to become the eventual standard for data networking over ATM.

Pick one of the methods in [15] for encapsulation of IP datagrams. Any one of the mechanisms identified should work. One will eventually become the standard method, but it is impossible to say which one, so the network engineer should go ahead and pick one and make sure it is used throughout the network.

Use directed ARP for address resolution. If equipment cannot be modified to use directed ARP, the best alternative in the near term is probably hand-coded address resolution tables.

Use ATM permanent virtual circuits (PVCs). PVCs provide the most similar service to the fixed networks in use today, so using them should necessitate fewer changes than using switched virtual circuits (SVCs). In any case, most ATM equipment will not provide SVC capability in the short term.

Use a performance-enhanced TCP. It is imperative to implement the mechanisms in [18, 19, 20] to ensure good performance on high-bandwith networks.

A Acronyms

AAL	ATM adaptation layer
ACK	acknowledgement
ANSI	American National Standards Institute
ARP	address resolution protocol
ATM	asynchronous transfer mode
B-ISDN	broadband ISDN
BBN	Bolt Beranek and Newman, Inc.
BOM	beginning of message
CBDS	connectionless broadband data service
CCITT	International Telegraph and Telephone Consultative Committee
CL-PDU	connectionless protocol data unit
CLNAP	connectionless network access protocol
CLNAP-PDU	connectionless network access protocol protocol data unit
CLP	cell loss priority
CLSF	connectionless service function
COM	continuation of message
CPU	central processing unit
CRC	cyclic redundancy check
CS	convergence sublayer
CS-PDU	convergence sublayer protocol data unit
EOM	end of message
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
Gb/s	gigabits per second
GFC	generic flow control
GOS	grade of service
HIPPI	High Performance Parallel Interface
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	integrated services digital network
ISO	International Organization for Standardization
LAN	local area network
LLC	Logical link control
MB/s	megabytes per second
Mb/s	megabits per second
MID	message identifier
MTU	maximum transmission unit
NLPID	network layer protocol ID
ms	milliseconds
NNI	network node interface
OAM	operation and maintenance
OC-3c	Optical Carrier Level 3 Concatenated
OC-12c	Optical Carrier Level 12 Concatenated
OSI	Open Systems Interconnect
PBX	private branch exchange
PC	personal computer
PCI	protocol connection identifier
PDN	public data network
PDU	protocol data unit
PTI	payload type indicator
PVC	permanent virtual circuit
RFC	request for comments
SAR	segmentation and reassembly
SAR-PDU	segmentation and reassembly protocol data unit
SAR-SN	segmentation and reassembly sequence number
SDU	service data unit
SEAL	simple and efficient adaptation layer
SMDS	Switched Multi-Megabit Data Service
SMTP	Simple Mail Transfer Protocol
SNAP	subnetwork access protocol
SONET	Synchronous Optical Network
SSM	single-segment message
SVC	switched virtual circuit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UNI	user–network interface
VBR	variable bit rate
VC	virtual channel
VCI	virtual channel identifier
VP	virtual path
VPI	virtual path identifier
WAN	wide area network

B CCITT Recommendations for B-ISDN

I.113 Vocabulary of terms for broadband aspects of ISDN
I.121 Broadband aspects of ISDN
I.150 B-ISDN ATM functional characteristics
I.211 B-ISDN service aspects
I.311 B-ISDN general network aspects
I.321 B-ISDN Protocol Reference Model and its application
I.327 B-ISDN functional architecture
I.361 B-ISDN ATM layer specification
I.362 B-ISDN ATM Adaptation Layer (AAL) functional description
I.363 B-ISDN ATM Adaptation Layer (AAL) specification
I.364 Support of broadband connectionless data service on B-ISDN
I.413 B-ISDN user-network interface
I.432 B-ISDN user-network interface—Physical Layer specification
I.610 OAM principles of B-ISDN access

References

[1] ANSI T1S1.5/91-449. *AAL 5—A New High Speed Data Transfer AAL.* ANSI, November 4–8 1991.

[2] ANSI Working draft proposed American National Standard for Information Systems. *High Performance Parallel Interface—Memory Interface.* ANSI, March 21, 1991.

[3] ANSI X3T9.3/88-023. *High Performance Parallel Interface—Mechanical, Electrical, and Signalling Protocol Specification.* Revision 8.0. ANSI, 1988.

[4] ANSI X3T9.3/89-013. *High Performance Parallel Interface—Framing Protocol.* Revision 3.1. ANSI, 1989.

[5] ANSI X3T9.3/90-119. *High Performance Parallel Interface—Encapsulation of ISO 8802-2 (IEEE Std 802.2) Logical Link Control Protocol Data Units,* Revision 2.0. ANSI, 1990.

[6] ANSI X3T9.3/91-023. *High Performance Parallel Interface—Physical Switch Control,* Revision 1.7. ANSI, 1991.

[7] Cáceres, Ramón. Peter B. Danzig, Sugih Jamin, and Danny J. Mitzel. "Characteristics of Wide-Area TCP/IP Conversations," *Computer Communication Review,* Vol. 21, No. 4, September 1991, pp. 101–112.

[8] Cáceres, Ramón. "Efficiency of ATM Networks in Transporting Wide-Area Data Traffic," *Computer Networks and ISDN Systems,* in process.

[9] Cooper, Eric. *Transmission of IP Datagrams over Asynchronous Transfer Mode (ATM) Networks.* DRAFT, November 1991.

[10] Cypher, David and Shukri Wakid. "Standardization for SONET and ATM and Open Issues," *Proceedings of the 3rd Annual Workshop on Very High Speed Networks,* Greenbelt, MD. March 9–10, 1992

[11] Escobar, Julio, and Craig Partridge. "A Proposed Segmentation and Reassembly (SAR) Protocol for use with Asynchronous Transfer Mode (ATM)", *Proceedings of the 2nd IFIP WG6.1/WG6.4 International Workshop on Protocols for High Speed Networks,* Stanford, CA, November 1990.

[12] Garrett, John, John Hagan, and Jeff Wong. *Directed ARP.* Internet Draft, November 17, 1991.

[13] Goldstein, Fred R. *Compatibility of BLINKBLT with the ATM Adaptation Layer.* ANSI, T1S1.5/90-009. February 5, 1990.

[14] Händel, Rainer, and Manfred N. Huber. *Integrated Broadband Networks: An Introduction to ATM-based Networking.* Addison-Wesley. 1991.

[15] Heinanen, Juha. *Multiprotocol Interconnect over ATM Adaptation Layer 5.* Internet Draft, October 16, 1992.

[16] IEEE Std 802.2. *Information Processing Systems—Local Area Networks—Part 2: Logical Link Control.* IEEE, December 1989.

[17] ISO/IEC TR 9577. *Information Technology—Telecommunications and*

Information Exchange Between Systems—Protocol Identification in the Network Layer. ISO, October 1990.

[18] Jacobson, V. "Congestion Avoidance and Control". *Computer Communication Review*, Vol. 18, No. 4, August 1988, pp. 314–329.

[19] Jacobson, V., R. Braden, and D. Borman. *TCP Extensions for High Performance.* RFC 1323, May 1992.

[20] Jacobson, V., and R. Braden. *TCP Extensions for Long-Delay Paths.* RFC 1072, October 1988.

[21] Lyon, T., F. Liaw, and A. Romanow. *Network Layer Architecture for ATM Networks.* DRAFT, June 26, 1992.

[22] Piscitello, Dave, and Joseph Lawrence. *The Transmission of IP Datagrams over the SMDS Service.* RFC 1209, March 1991.

[23] Renwick, John K., and Andy Nicholson. *IP and ARP on HIPPI.* Internet Draft, January 1992.

[24] RFC 791. *Internet Protocol.* September 1981.

[25] SR-NWT-001763. *Preliminary Report on Broadband ISDN Transfer Protocols*, Issue 1. Bellcore, December 1990.

[26] Tsuchiya, P. *Shortcut Routing: Discovery and Routing over Large Public Data Networks.* Internet Draft, July 1992.

Appendix B

Spengler, Michael K. and Salo, Timothy J., *Technologies for Gigabit Distributed File Systems*, Minnesota Supercomputer Center, Inc., 1992

This deliverable is referred to at "Analysis of Remote File System Requirements and Technology" in the VHSRFS proposal.

Technologies for Gigabit Distributed File Systems

Michael K. Spengler
Timothy J. Salo
Minnesota Supercomputer Center, Inc.*

December 14, 1992

Abstract

Gigabit-per-second wide-area networks will soon be moving out of the research laboratories. A model for one use of these networks, the network supercomputer, is presented. A key component of the network supercomputer is its data sharing service. This paper discusses the requirements for such a service and provides an analysis of the current technologies available for its implementation.

1 Introduction

The commercial availability of a new generation of high-speed networks is imminent. Both local- and wide-area gigabit-per-second networks are already appearing in research networks and national testbed projects. This considerable increase in network bandwidth provides users with opportunities to explore new classes of problems and allows the use of new, innovative techniques for solving a wide range of current problems.

An especially interesting area that is emerging as a viable technology is distributed, or collaborative, computing. This is a system in which a group of (potentially widely dispersed) computers all participate in the solution of a problem. Each computer is often devoted to solving a particular portion of the problem or to providing a particular service to the other computers in the computing group. One such model of computing, the network supercomputer, is introduced in Section 2.1.

*The research described herein was sponsored by DARPA through the U. S. Army Research Office, Department of the Army, contract number DAAL03-91-C-0049. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by either DARPA or the U. S. Army.

One of the key elements in distributed computing is the ability to efficiently access and share large amounts of geographically-distributed data. The performance of such a data sharing service is a major factor in the success of a distributed computing system.

This paper presents a set of requirements to guide the design and implementation of a data sharing service (see also [23]). Some current research in this area is discussed and some of the technologies available for constructing such a service are presented. Finally, an overview of a few well-known or important commercial and research systems is given along with a discussion of their suitability in the network supercomputer model.

2 The Model

2.1 The Network Supercomputer

The *network supercomputer* is a high-performance computing system which is comprised of geographically distributed components communicating with each other via a gigabit network.

The components of high-performance computing systems have, until now, been collocated in the interest of overall system performance. The speed of the channels connecting components of a supercomputer system (for example, the channels connecting mass storage to the processor) has a great effect on the overall performance of the system. Collocation was mandated by the order-of-magnitude differences in speed between computer channels and wide-area networks (e.g. compare an 800 Mb/s HIPPI channel to a 1.5 Mb/s T1 telephone line, or even a 45 Mb/s T3 line). With

experimental gigabit per second networks under construction, geographic distribution of the components of a supercomputer system is no longer impractical.

A network supercomputer might be composed of a number of high-performance components communicating via a gigabit network. Some or all of the following components might be configured into a network supercomputer:

- A vector supercomputer,
- A massively parallel supercomputer,
- A shared, very high-speed remote file system,
- A high-resolution graphical display,
- A source of real-time data, perhaps at very high speed.

A network supercomputer would be "constructed" by using a gigabit network to connect the various components together. When the network supercomputer was no longer needed, the connections would be dropped, and it would cease to "exist". A researcher could construct network supercomputers for a variety of purposes, including:

- To create a supercomputer using a prototype or one-of-a-kind machine,
- To rapidly create a prototype of a mixed-architecture system,
- To access a very large or time-critical data base,
- To integrate data from multiple archives.

Most of the components of the network supercomputer are now available, or are under construction. Both vector and massively parallel supercomputers are available, and they already have network interfaces operating at high speed (e.g. HIPPI interfaces at 800 Mb/s). Experimental gigabit networks are being constructed, and are the focus of much research.

The component that remains to be studied and constructed is the very high-speed remote file system.

2.2 File Systems

In order to meet some of the objectives of the network supercomputer model, researchers throughout the (inter-)network must be able to gain access to the available data repositories. Similarly, the "owners" of these repositories must have the means to make this data accessible to the network public and to control the access to their data. To fully implement a true data sharing service requires a complete (and complex) set of protocols—namely a Distributed File System (DFS).

Some of the features of a DFS which would make it highly suitable for the data sharing component of our model include:

Distributed architecture: One of the distinguishing characteristics of a DFS is, appropriately, the distribution of functionality between the data accessor (client) and the data provider (server). The server is responsible for maintaining the integrity of the underlying file system data and for managing the client's access to that data. The client makes requests for data from the DFS server in much the same way as it would to its local native file systems, though the client must contain the necessary DFS protocol software for interfacing with the server. The complexity of the DFS protocol(s) can vary widely, depending on many architectural design decisions in the DFS.

Host Independence: The clients and servers are typically (though not necessarily) resident on separate hosts, whose only dependencies are the availability of network connections between themselves. The servers and clients are independently administered and each may set its own performance criteria and security policies.

Wide Availability: DFSs are based on the principle that the clients and servers are connected via networks, thus making the data repositories accessible to any client with network access to the server's network. Most DFSs do not explicitly distinguish between LAN and WAN interconnections.

Shared access: Since one of the key elements of the network supercomputer model is widespread sharing of research data, the

ability for multiple users to simultaneously share repository data is crucial. All of the DFSs discussed in this paper provide support for simultaneous read access for multiple clients. However, support for multiple writers/updaters varies among the DFSs.

Performance: Since DFSs are specifically designed to provide remote file service, they have the potential to operate at very high performance levels. However, there are many design attributes that greatly affect the various performance aspects, as will be discussed in the next sections.

Data access patterns can have a large impact on file system performance. Although difficult to predict, a few points seem like reasonable assumptions:

- Typical files will be large and dynamic (hundreds of MBytes).
- Each client will only reference a small number of files during a given connection.
- Each file will only be opened once per connection.

Most currently used shared files have little or no internal structure, i.e. they are merely a sequential collection of data records. To utilize such a file, a program must sequentially search through the data until the desired information is located. In the network supercomputer model, however, it is envisioned that these large shared data repositories will move towards a more database-oriented structure, with a more flexible and efficient data organization and access method. With this trend in file structures, the following access patterns may begin to emerge:

- Client accesses will be mostly reads, with some update.
- Although sequential file access will continue to predominate, random file access will expand in usage.

2.3 Networks

The networking environment envisioned in the network supercomputer model will include a relatively small range of configurations. Since the focus of the model is upon widespread sharing of information, most networks will cover large geographic areas. Both public backbone networks, such as the Internet and the National Research and Education Network (NREN), and wide-area research networks and testbeds are examples of networks which will be used in this environment. Some of the common features of these networks are likely to be:

- High-speed (100Mb/s to >1Gb/s),
- Relatively long round-trip times (RTT) as compared with LAN environments (>30ms for US transcontinental fiber),
- Support for the Internet Protocol (IP) family.

While there is still much work to be done, the hardware and software required to build these networks are mostly in hand. However, the major challenges ahead lie at the network periphery—host hardware interfaces, operating system software structures and interfaces, and network protocols all loom as bottlenecks to the efficient utilization of these new network environments. Finally, the file system protocols themselves must be designed to take advantage of these new performance opportunities.

3 DFS Requirements

This section discusses some of the features that are important for a DFS implementation to adequately function in a network supercomputer environment. Some performance considerations and requirements are also presented. In order for a DFS to take advantage of the facilities available in a network supercomputer, performance issues must be addressed. Lastly, some current design issues in DFSs are presented, along with some discussion on their impact on our DFS requirements.

3.1 Feature Requirements

There is a wide range of capabilities and characteristics which may be, in general, attributed to distributed file systems. We present here those which we believe to be of the most importance to a network supercomputer DFS implementation.

Networking: Any DFS, by definition, must include support for some form of networking. However, our model places some additional requirements beyond a simple networking implementation. The DFS must be capable of running over standard networking protocols, i.e. it must not require a proprietary or specialized networking layer. Specifically, the DFS must be able to operate over Internet Protocol (IP) networks, since it is envisioned that most network supercomputer networks will be IP based or at least will support IP. The DFS protocols themselves should be independent of, or capable of adapting to, a large variety of network connections with varying performance characteristics, e.g. high-speed LANs and high-speed, high-delay WANs. The design of the DFS should not prevent the utilization of new networking technologies and protocols, in particular any IP follow-on networks, when they become available.

Scalable: In order for a network supercomputer DFS to be successful, it must be designed to scale well as increasing demands are placed on it. Among the many dimensions of scaling [11], two are especially important for a DFS.

As the number of users and clients grows, the DFS must maintain a high level of performance. Since ever-growing demands will eventually exceed the capacity for a single system, the DFS should be able to distribute, transparently to the user, the server functionality across multiple systems.

A network supercomputer DFS must also be capable of operating over a wide range of capacities. Both large file sizes (over 1 GByte) and large file system and repository sizes (tens of TBytes) must be efficiently supported. A hierarchical storage management system, which automatically migrates data between storage media with varying levels of performance and cost, will make such scaling requirements more technically and economically feasible.

File system semantics: The file system semantics presented by a DFS are important to the users of a DFS. The behavior of the systems servicing requests must be clear and consistent, independent of object locations and failure modes. One particular semantic model, familiar to a large percentage of supercomputer users, is the UNIX[1] file system's semantics. A DFS supporting strict UNIX file system semantics would be welcomed by many users.

Portability: A DFS server should be accessible from a large variety of client hosts and operating systems. To encourage this widespread availability, the DFS client should be readily portable, with few, if any, operating system modifications. The DFS server, however, is designed for more specialized functionality, so that some portability features will probably be sacrificed. It should, however, be designed to allow it to take advantage of new technologies.

User Program Interface: In order to promote user program portability, access to the DFS client facilities should be through the native system calls and libraries functions. The client implementation would reside below the user-accessible interfaces. Note that in many operating system environments, this structure would violate the desire to avoid operating system modifications in the DFS client implementation.

File Types: The DFS system must not assume any structure or content to the data files processed. The files must be treated as *bitfiles* [5]—a string of bits unconstrained by size or structure. The software layers above the DFS are responsible for the interpretation of a data file's contents.

Transparency: The ability for processes to execute, unchanged and unaware, on any node in the network supercomputer is crucial. Thus, many details of a DFS should be transparent to users. For example, if a system preserves location transparency, processes do not know

[1] UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

where a resource is located. Access transparency provides that local and remotely located resources are accessed in the same way. Other types of transparencies relate to failure conditions, name space, concurrency control, and resource replication [2].

Administrative control: The flexible and effective administration of a file system is also an important feature of a DFS. Providing various levels of security, access control and authentication is especially important in systems spanning large, publicly-accessible networks. The ability to dynamically control and manage file system resources is crucial for maintaining system responsiveness and high levels of availability.

Client access: Although the DFS may utilize its own client-server protocol, support for other commonly used protocols should be provided. This would allow systems without DFS support to gain access to the DFS repository, although at potentially lower performance levels and with fewer features supported. The DFS should, at least, allow for FTP and NFS access.

3.2 Performance Requirements

A network supercomputer operates in a very high-performance environment. In order for the system to deliver its desired high level of performance, each of its components must provide their own services at a high enough level of performance to ensure that they do not become a system performance bottleneck. Some requirements and considerations are presented here to ensure that the network supercomputer DFS is able to deliver high-performance file service.

System Hardware: When designing and selecting system hardware for a DFS server, there are obviously many decisions to make to ensure a high performance platform is available. In order to rapidly move and process such large quantities of data, the file server must be based on a very powerful processor containing large amounts of fast memory — comparable to today's supercomputers (without needing the vector processing capabilities). The large memory capacities are necessary for file system caching and the network protocol (re)transmission queues required for operation over high-speed, high-delay networks. In addition, high-performance disk systems and network I/O controllers must be available to get all of this data in and out of the system.

Another important facet of system performance is interrupt processing. Typically, every network packet received or sent on an interface generates an interrupt on the host machine. To achieve high levels of networking performance, either the packet sizes must be kept large (thus reducing the number of interrupts to be processed) or the host must possess high interrupt density processing capabilities. Given the current (and near-term future) state of networking protocols, a large percentage of small packets will remain on the networks [4]. In addition, relative to their raw data processing power, interrupt processing power is a weak link in many of today's supercomputers.

System Software: While the choice of DFS server system software is based on many factors, there are some performance-related issues to be considered. The networking software implementation is certainly one area of concern. The efficiency of the networking software and its interfaces will be a key factor in obtaining an overall high level of server performance.

Since a file server is mainly involved in moving large amounts of data between the storage systems and the network, efficient internal management of data is crucial. The networking protocol software and its user-level interfaces must strive to keep data copying to a minimum, for this can become a major performance bottleneck when using high-speed networks.

Multi-threading support, which is a system service that allows designers to easily and efficiently build concurrent applications and which can also provide transparent multiprocessing support, may give a DFS server a performance boost. By structuring the DFS software to take advantage of multi-threading, the server may be able to more efficiently deal with larger numbers of client connections and outstanding network I/O requests without incurring large operating system overheads such as context switching.

Since a DFS server's main function is to provide file service, the availability of high-performance native file systems is crucial. In addition, the operating system should provide an efficient high-level file system abstraction layer to allow the DFS designers to easily design and implement a custom file system, if required.

Network Protocols: Although the choice of networking protocols is intimately tied to the design of the DFS protocol (see below), it will have a large impact on the overall performance limits of a DFS server. If the networking protocols can't deliver the data fast enough, it doesn't matter how fast the DFS server itself may be. Among the considerations for networking protocols are the types of services provided:

- Connectionless (datagram) or connection-oriented,
- Record delineated data transfer or byte streams,
- Service delivery guarantees, such as reliable delivery and prioritized message delivery,
- Availability of remote IPC services.

The protocols must also be able to perform well over a wide range of network technologies and topologies, from LAN environments to high-speed, high-delay WAN connections, where the amount of data "in flight" can be in the mega-byte range. Another, more practical, consideration is how standardized, widespread and available is the protocol. If there are few implementations and deployments of a protocol, or just as bad, if the protocol's implementations aren't interoperable, there will be limitations placed on the ability of clients to interact with the server, especially for users attempting connections through the many standard public data networks.

DFS Protocols: The DFS protocols are those used between the DFS servers and their clients, and are responsible for the management of all external DFS functions. As mentioned earlier, DFS protocol decisions are greatly influenced by the availability of many operating system and networking capabilities. For example, in determining which connection-oriented protocols to use, the designer must consider the expected longevity and number of packets for the connection. Since there is (typically) connection setup and teardown overhead associated with connection-oriented protocols, it may make more sense to use a connectionless protocol if there are apt to be a large number of short-duration sessions and a connection-oriented protocol for large bulk-data transfers. Further treatment of some of the issues can be found in Section 3.3.

3.3 Design Issues

Some of the current topics related to distributed file system design are presented, along with some analysis of the implications of these design decisions and how they will affect a potential network supercomputer DFS.

Server State: In a distributed file system, there is a large amount of state information which is distributed among the clients and servers in the network. Such information as the status of all hosts, outstanding file system requests and cache conditions must be maintained. The approach taken to manage this information is pivotal for the overall system design.

In the *stateless server* approach, the server operates independently of its clients and maintains no state information about previous client requests. Any "permanent" state information, such as any file system modifications, must be written to disk before the server can complete processing of the affected request. The main advantage of the stateless server model is the ease of recovery from server crashes. Since no state information is lost during the server crash, there are no special recovery protocols required after the server is restored to service. The client merely keeps retrying any outstanding operations to the server—eventually, when the server is operational again, the request will be processed and the client will receive the response.

Since the client of a stateless server is unaware of any file system state modifications, however, the client must continually poll the server to verify its internal view of the file system state. This constant need for polling

is wasteful of client, server and network resources. Any client caching policies are also adversely affected, since there are now no consistency guarantees possible—that is, the client can never know if its cached information is valid and up-to-date.

Stateful servers, however, follow a more optimistic approach. The server will maintain volatile information, typically in process memory, about the file system, its clients and how its resources are being utilized. The server will inform the client whenever something of interest to the client has changed. This extra state information allows the server greater functionality and gives it the ability to optimize the server's (and client's) resources. For example, a stateful server can support strict UNIX file semantics, such as **unlink**ing an open file, and all forms of file locking (file, byte range, etc), whereas a stateless server is unable to. In addition, with a stateful server, client-server authentication only needs to be performed once per connection, while a stateless server must authenticate each transaction, placing extra processing demands on the server. The Spritely NFS project [19] showed that replacing the stateless server in NFS with a stateful server significantly improved overall file system performance.

In a stateless server system, file system writes are performed synchronously by the server—the client must wait until the server has physically written the data to disk and returned the response. This causes write performance to be limited by disk and network speeds and by network propagation delays. On the other hand, stateful servers allow file system writes to be cached in memory (either client or server memory) and only later is the cached data written to disk. Client system performance can thus be drastically improved.

Caching Strategy: The implementation of a well-designed caching system can lead to dramatic performance improvements for a distributed file system. Virtually every modern DFS now uses some form of file system caching. There are two separate forms of caching: server caching and client caching.

Server caching is, as suggested by its name, performed on the file system servers. Recent file references (both data and directory information) are cached in the server's main memory, such that subsequent requests for this information will be satisfied from the cached copy, thus avoiding the overhead of disk access.

On the other side of the network, *client caching* involves the caching of remote (server) file system information at the client node. Client-side caches are maintained in the client's main memory and also, for some DFS implementations, on the client's local disk. Effective client caching can avoid the overhead and latency of network I/O and server disk accesses. The design of an effective client caching strategy is, as mentioned earlier, closely associated with a stateful server DFS design. The combination of both forms of caching can, in addition to the performance gains, improve the scalability of a DFS by greatly reducing both network and server loads.

One distinguishing feature of cache designs is the granularity of the cached units of data. Some systems, such as the Amoeba Bullet file server [24], only cache entire files. While simplifying certain implementation issues, this severely, and for us unacceptably, limits the maximum file size allowed and, for very large, sparsely accessed files, dramatically and unnecessarily increases network load. Other caches are typically organized with fixed-sized file blocks, in which the client only retrieves and caches those portions of a file which are actually accessed. This design eliminates the file size limits, while giving better performance and utilization for sparsely accessed files without incurring any significant penalties when the entire file is processed.

With the introduction of client caching, however, comes the problem of how to keep multiple, cached copies of file data up to date with changes made in remote copies of the same data. This problem, known as maintaining *cache consistency*, is probably the most crucial design element for client caching systems. The ideal goal of cache consistency management is to provide the user with single-system file semantics, regardless of the location and number of simultaneous data users. Note that consistency is not the same as correct synchronization—users must still synchronize their actions to ensure that their file operations are performed in a sensible order. There are various levels of consistency guarantees provided by past and current DFS systems.

Some use timer-based mechanisms which allow small fixed maximum windows of inconsistency, or "stale" data, to exist, while others implement more elaborate client-server protocols, such as using callbacks and tokens, to provide stronger consistency guarantees. Some specific implementations of cache consistency strategies are mentioned in Section 4.

The handling of file block writes is also an important design element for both client and server caching performance. A *write-through cache* design requires that the data be written to the server or disk before the write operation completes. While adding some extra measure of reliability, this design greatly increases write operation latencies, as opposed to *write-back* or *delayed-write* designs which merely write the data into the cache and write the data to the disk or server after the file has been closed.

Replication Strategy: In order to provide higher degrees of file system availability, some DFS implementations use various methods of file replication, wherein copies of files are stored at multiple servers. Clients are then able to access the file data as long as they can contact any of the servers. Some systems only support read-only replication of data, while other, more recent DFSs, provide for read-write replication of data [18]. Obviously, server consistency and conflict resolution mechanisms must be provided to recover from server crashes and network partitionings. An optimistic strategy will allow updates to occur in all partitions, while a pessimistic one only allows updates in one partition. A further complementary enhancement to a replication strategy is a client's ability to run in disconnected operation mode, which occurs when a client is unable to contact any servers for a file. In this mode, the client will only use its cached copy of the file, resuming its normal operation mode as soon as it is able to reconnect with a file server.

Native File Systems: The native host file systems provided on file servers will also have an impact on the performance and availability of the DFS file servers. The local disk allocation strategies, such as block-based, contiguous and extent-based, can also influence the DFS's caching system implementation. New physical file systems are also currently being researched. For example, *log-based file systems*, which are derived from database integrity management systems, maintain transaction logs for the file system, which allows them to provide a rapidly recoverable and available file system after system crashes/restarts. Other features, such as logical volume control and integrated access-control capabilities, are also capable of being exploited by a DFS server.

Networking: Distributed operating systems and distributed file systems often provide their own network and transport layers. The perception seems to exist that "standard" protocols, such as IP and TCP, don't provide the features and/or performance required for their systems. The use of standard protocols would enable more rapid system implementation, ease porting efforts and allow a higher degree of interoperability between distributed systems. The system designers and developers should be able to devote their resources towards the problems at hand, rather than being forced into developing an entire ad hoc networking environment as well.

Since, as discussed in Section 3.1, the IP family of protocols is of primary concern, there should be an effort within the IP community to address the real concerns of the distributed computing environment community. Recent work [6][12] has shown that the IP and TCP protocols themselves are not impediments to high-performance networking, but rather that the implementations and hardware bandwidth limitations are the major sources of bottlenecks.

The most critical needs are in the transport layer, especially in supporting transaction processing requirements. Nearly all modern distributed systems rely upon RPCs for their communications services (see below). However, neither of the transport-layer IP protocols (TCP and UDP) provides all of the services and features which an RPC protocol requires. In an attempt to stimulate discussion leading to the development of such a new transport protocol, Braden [3] presents a good introduction to the services which this protocol should provide. Indeed, there are probably even more requirements today to add to its recommendations. There are protocols with some potential, such as VMTP, RDP and XTP, but none of these has all of

the required features, nor has any progressed to the Internet standards track yet.

Ideally, any required transport (or network) layer service should be available to applications, such as a DFS or an RPC system, merely through the passing of parameter lists. Such options as connection-oriented vs. connectionless communications, reliable delivery of data, multiple outstanding datagram requests, high-speed bulk data delivery and prioritized data delivery should all be possible in any desired combination though one (or more) standard transport providers.

Remote Procedure Call (RPC):

For distributed systems, the ability to set up and manage network communications between the nodes in the system is obviously essential. However, the complexity of managing multiple communication channels over diverse network protocols can be substantial. Distributed system designers and implementors would benefit from a single communications interface which is isolated from and independent of the details of the specific underlying networking systems. Most modern distributed systems are based on client-server paradigms utilizing remote procedure calls (RPCs) [1] as the preferred communication methodology. An RPC facility generalizes the familiar local procedure call capability to also include "procedure calls" to processes running on remote, network-connected systems. Details such as data encoding formats, transport protocol requirements, etc. are transparent to the RPC user, thus providing a simple, familiar programming model. The actual location of the requested service is also transparent to the user, including whether it is located locally or on a remote node. This model enables a clean, modular design and a straightforward implementation for a wide variety of network services and servers [13].

RPC invocations often consist of request-response transactions, which are typical in client-server interactions. Many RPC systems only allow one request to be outstanding (i.e. pending a response acknowledgement) from any application, blocking the application until the response is received from the remote node. In a LAN environment, this doesn't present many problems since the network latency is very small. However, when communicating across a high-speed, wide-area network with (relatively) large round-trip delays, the effects of this policy can be drastic. The application blockage can be viewed as equivalent to lost bandwidth (as well as lost processing), since the network, at least from the application's viewpoint, cannot be utilized. There are (at least) two solutions to this problem. The RPC facility may provide *asynchronous RPCs*, in which the RPC system returns control immediately to the application after accepting the call and its parameters. The caller is also responsible for providing (with the RPC call) a pointer to a *callback* routine for each request. When the response is finally received, the system will give control to the callback routine, and pass it the results (returned from the server in the response) of the RPC transaction. Another approach to solving the "lost bandwidth" problem is through the use of multi-threading. Using this method, the client creates a new thread to handle each RPC request, thus allowing other client threads to continue their processing while the thread is blocked by the RPC call. A server implementation utilizing multiple threads of control might also realize similar, or even greater, performance gains.

Although there are two widely available commercial RPC systems currently in use (Sun/ONC RPC [21] and NCS/DCE RPC [9][16]), a standardized RPC system is not likely in the near-term. Current RPC systems often are forced to include such functionality as datagram reliability, reliable multicasting, meaningful and accurate round-trip timings, congestion management, flow control and message fragmentation and reassembly. An expanded set of standardized transport protocols, as mentioned above, is required before RPC systems can be standardized, since the necessary transport services will not be available otherwise and the RPC system is not the proper location for such functionality.

There are a number of features that a standardized RPC system should provide. For performance, it should be able to fully utilize any high-speed network connections and provide low-latency response (independent of network latencies) RPC calls. It should also scale well with increasing numbers of users and requests and should be able to perform well in multi-processing environments.

As mentioned above, support for both asynchronous RPCs and multi-threading environments should be available. For applications such as distributed file systems, the RPC service should provide a mechanism for efficient bulk data transfers between the client and server. For many installations, data security is a large concern—the RPC facility should provide a secure yet flexible communications facility. Finally, the RPC service should allow both user-to-user and kernel-to-kernel RPC communications. Distributed operating systems, as well as other kernel-based networking services, must have the same functionality provided to user-level services.

4 DFS Implementations

This section will present a few of the more well-known or important implementations of Distributed File Systems and will discuss some of the main features of each system. An analysis of the DFS's ability to meet the previous section's requirements will also be given.

One class of systems not represented here is systems commonly classified as mass storage systems. These systems have been designed mostly for a local computer center environment and are mainly concerned with such issues as hierarchical storage management and the use of new storage media and devices. These systems often require the user to provide manual or semi-automated control for file access. The file access/transfer protocols used are usually just FTP or, occasionally, NFS. They typically do not allow for the shared, multi-user access that is required in the network supercomputer model. So, while these systems are important in their own right, they will not be further considered here.

4.1 NFS

The Network File System (NFS) [20] was introduced by Sun Microsystems, Inc. in 1984 to provide a simple and transparent filesharing service for nodes in a heterogeneous network of machines and operating systems. NFS relies upon Sun/ONC RPC [21] for its communications services and uses the External Data Representation (XDR) specification [22] to achieve data sharing among machines of different CPU architectures. NFS is probably the most widely used and implemented DFS and has become a de facto standard for file sharing services.

To achieve its goal of simplicity, NFS is designed with a stateless server protocol. As a result (see Section 3.3), NFS is not able to support all of the UNIX file system semantics, such as removing open files. Some functionality, such as file locking support, is actually implemented as a separate, non-integral part of the file service. With a stateless protocol, NFS can provide no guarantees of file system consistency. Clients must periodically poll the server to maintain consistent cache information, though this obviously creates the possibilities of windows of inconsistency between polling operations and also adds to network and server processing overhead. To help increase file system performance, NFS does provide some client-side caching support and uses read-ahead to boost read performance. However, NFS suffers from poor write performance since writes are synchronous at the server node, resulting in high latency for each write request. NFS performance also can suffer from its 8 KB maximum data block limit, especially across high-performance networks with large MTUs and high per-packet overhead, such as HIPPI. Lastly, the Spritely NFS project has shown that, by adding a stateful cache consistency protocol to NFS, file system performance can be significantly improved.

Even though there are some significant feature and performance drawbacks to NFS, its popularity is not to be overlooked. Thus, any new DFS system seeking to establish itself in the market would be wise to provide access to NFS clients, though obviously with much lower performance characteristics than with native client access.

4.2 RFS

The Remote File Sharing (RFS) system [8] was introduced by AT&T in 1986 for the UNIX System V operating system. All RFS communications are implemented as a special protocol via the standard Transport Layer Interface (TLI). RFS is, in actuality, a distributed UNIX file system, since all RFS calls are implemented as remote UNIX (System V) system calls—the client ker-

nel calls are merely extended across the network. This does, however, allow RFS to support 100% of the UNIX file system semantics. RFS does maintain a stateful server protocol, guaranteeing consistency, but does not recover server state following a server crash—current client connections are merely forcefully terminated. Since all client requests are directly serviced across the network by the file server, file operations are delayed by network and server latencies and performance suffers.

Although RFS is an effective UNIX System V file service, its scope is too narrow and capabilities too limited to warrant consideration as the basis for a DFS for the network supercomputer model.

4.3 AFS/DCE

The Andrew File System (AFS) [18] was developed at Carnegie Mellon University beginning in 1982. AFS 2.0 was the starting point for the CMU Coda file system [18], which strives to maintain a highly available file system through server replication and *disconnected operation*. In 1990, AFS 4.0 was selected by the Open Software Foundation (OSF) as the basis for the file system component of its Distributed Computing Environment (DCE) [14][7]. The discussion here will focus on the DCE implementation of AFS.

The DCE DFS uses a stateful server protocol and is designed to support a large, diverse computing environment. Its communication services are provided by the NCS 2.0 RPC package [15], which supports both datagram and connection-oriented streaming services. There is a heavy emphasis on client caching, with both client memory and local disk caching supported. The DFS uses a set of typed *tokens* for cache consistency management, in which the tokens represent various types of guarantees or rights granted by file servers (e.g. data access, locks, status modification). This system allows the DFS to support strict single-system UNIX (and POSIX) file system semantics, while at the same time reducing the network overhead required to maintain the file system consistency. The use of large 64 KB cache file chunks enhances performance by reducing both server and network loads.

File system availability is enhanced via the replication of file system meta-data (e.g. file location information, directory information) and automatic read-only file systems replication. This DFS also contains the Episode log-based physical file system, which provides fast restart capabilities as well as high performance. Good support for a server's existing local physical file systems is also provided. File system security is accomplished through per-file access control lists and authentication protocols. To provide enhanced interoperability, the DFS is also able to export its file systems via the NFS protocol.

With the adoption of AFS 4.0 by the OSF DCE, the potential now exists for a large number of vendors producing interoperating versions of this DFS. Along with its many features and high-performance capabilities, the DCE DFS warrants serious consideration as the data sharing service for a network supercomputer model.

4.4 Sprite

Sprite [17] is an experimental distributed operating system under development since the mid-1980's at the University of California at Berkeley. It is designed to support high-performance, networked engineering workstations in a moderate sized environment. Much of the system has been modeled to provide compatibility with BSD UNIX, including the Sprite monolithic kernel design. Its stateful server design and its use of the file system name space as a name service [25] are among its distinguishing contributions to distributed file system design.

Dynamically-sized client and server memories are used for file caching, with 4 KB cache data units. A Sprite-specific RPC facility has been developed to provide kernel-to-kernel communications with a maximum data message fragment size of 16 KB. This facility is heavily used by the Sprite kernel and its services. The Sprite file system consistency guarantees are designed so that file reads always return the most up-to-date data, regardless of how the file is being used in the network. For file write consistency, Sprite provides "sequential write-sharing", which uses file version numbers when files are opened to ensure that all cached file blocks are current. If more than one process is concurrently accessing a file, with at least one attempting to write to it, then the "concurrent write-sharing" mode will be invoked, in

which case all client caching for that file will be disabled, thus providing server-guaranteed consistency. Sprite studies have shown that such occurrences are of a low enough percentage that file system performance does not suffer much degradation. Write performance is also enhanced through the use of "delayed-write" operation—blocks are initially only written to the cache copy. Unless concurrent write-sharing is previously invoked, the data will only be written back to the server after 30 seconds.

The Sprite system has generated many important research results in distributed systems. However, due to its specific kernel requirements and communications requirements, it is not currently capable of making the transition to a easily ported, viable, widespread production system.

4.5 Amoeba

The Amoeba distributed operating system [10] was initially developed during the mid-1980's at the Free University of Amsterdam. It is designed to be an object-based, high-performance, transparent distributed system. It utilizes a microkernel architecture, with user-space server processes providing access to the system's various objects. The Bullet server [24] is Amoeba's innovative, high-performance file service. This DFS uses the concepts of contiguous file storage and immutable files to achieve high throughput and low delay in client file accesses. All files are immutable—once created, the file cannot be modified. To update a file, a new file is created which contains the updated data. Thus, files are stored as sequences of versions.

Caching is only performed on the file server, not clients. The system will transparently replicate entire files onto the client system, from where all client file access operations will be satisfied. While this enables the Bullet file system to attain high performance levels, it also exposes several deficiencies, which in the network supercomputer model are quite serious. The major problem is that the file server requires that the entire file fit into the client's memory-based cache—an unacceptable constraint. Since new versions of files are are constantly being created and then transferred in full to the clients, network loads may become high. There is on-going research into providing reasonable solutions to these problems.

As with the Sprite system, Amoeba can only be considered as a vehicle for research into distributed systems.

5 Conclusions

The coming availability of gigabit-per-second local- and wide-area networks will drive the need for distributed file systems in a network supercomputer environment. The area of distributed systems, including DFSs, is one with much current research in progress. Some areas in which research (and implementation) experience is needed are:

- Standardized network transport software capable of delivering high-performance, high-functionality services for RPC systems.
- Network interface hardware and software capable of performing at the full bandwidth levels of the emerging gigabit networks.
- File system protocols capable of high-performance operation over high-bandwidth, high-delay networks.

In the near-term, the most-promising prospect for a network supercomputer distributed file system appears to be the OSF DCE file service, which is based on the lastest version of AFS.

References

[1] Birrell, A. D. and B. J. Nelson. "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems*, Vol. 2, No. 1, February 1984, pp. 39-59.

[2] Borghoff, Uwe H. *Distributed File/Operating Systems.* Springer-Verlag. 1992.

[3] Braden, R. *Towards a Transport Service for Transaction Processing Applications.* RFC 955, September 1985.

[4] Cáceres, Ramón. "Efficiency of ATM Networks in Transporting Wide-Area Data Traffic," *Computer Networks and ISDN Systems*, submitted.

[5] Coleman, Sam, and Steve Miller, Editors. *Mass Storage System Reference Model.* IEEE, Version 4, May 1990.

[6] Jacobson, V., R. Braden and D. Borman. *TCP Extensions for High Performance.* RFC 1323, May 1992.

[7] Kazar, Michael L., Bruce W. Leverett, Owen T. Anderson, Vasilis Apostolides, Beth A. Bottos, Sailesh Chutani, Craig F. Everhart, W. Anthony Mason, Shu-Tsui Tu and Edward R. Zayas. "DEcorum File System Architectural Overview," *Proc. USENIX Conference Summer 1990*, June 1990.

[8] Kochan, Stephen G. and Patrick H. Wood, Editors. *UNIX Networking.* Hayden Books. 1989.

[9] Kong, Mike, Terence H. Dineen, Paul J. Leach, Elizabeth A. Martin, Nathaniel W. Mishkin, Joseph N. Pato and Geoffrey L. Wyant. *Network Computing System Reference Manual.* Prentice Hall. 1990.

[10] Mullender, Sape J., Guido van Rossum, Andrew S. Tanenbaum, Robbert van Renesse and Hans van Staveren. "Amoeba: A Distributed Operating System for the 1990s," *IEEE Computer*, Vol. 23, No. 5, May 1990, pp. 44-53.

[11] Neuman, B. Clifford. "Scale in Distributed Systems," *Advances in Distributed Computing: Concepts and Design*, IEEE Computer Society Press, 1992.

[12] Nicholson, Andy, Joe Golio, David A. Borman, Jeff Young and Wayne Roiger. "High Speed Networking at Cray Research," *Computer Communication Review*, Vol. 21, No. 1, January 1991, pp. 99-110.

[13] Notkin, David, Andrew P. Black, Edward D. Lazowska, Henry M. Levy, Jan Sanislo and John Zahorjan. "Interconnecting Heterogeneous Computer Systems," *Communications of the ACM*, Vol. 31, No. 3, March 1988, pp. 258-273.

[14] *Distributed Computing Environment.* Open Software Foundation, Report OSF-DCE-RD-1090-3, April 1991.

[15] *OSF Distributed Computing Environment Rationale.* Open Software Foundation, Report OSF-DCE-RD-590-2, May 14 1990.

[16] *Remote Procedure Call in a Distributed Computing Environment.* Open Software Foundation, Report OSF-O-WP10-1090-2, August 1991.

[17] Ousterhout, John K., Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson and Brent B. Welch. "The Sprite Network Operating System," *IEEE Computer*, Vol. 21, No. 2, February 1988, pp. 23-36.

[18] Satyanarayanan, Mahadev. "Scalable, Secure, and Highly Available Distributed File Access," *IEEE Computer*, Vol. 23, No. 5, May 1990, pp. 9-21.

[19] Srinivasan, V. and Jeffrey C. Mogul. *Spritely NFS: Implementation and Performance of Cache-Consistency Protocols.* DEC Western Research Lab., Research Report 89/5, May 1989.

[20] Sun Microsystems, Inc. *NFS: Network File System Protocol Specification.* RFC 1094, March 1989.

[21] Sun Microsystems, Inc. *RPC: Remote Procedure Call Protocol Specification: Version 2.* RFC 1057, June 1988.

[22] Sun Microsystems, Inc. *XDR: External Data Representation Standard.* RFC 1014, June 1987.

[23] Thomas, Joseph P., John David Cavanaugh, Timothy J. Salo. *Design of a Very High-Speed Remote File System.* Minnesota Supercomputer Center, Inc., December 1992.

[24] van Renesse, Robbert, Andrew S. Tanenbaum, and Annita Wilschut. "The Design of a High-Performance File Server," *Proc. 9th IEEE Int. Conf. on Distributed Computing Systems*, June 1989.

[25] Welch, Brent B. *Naming, State Management, and User-Level Extensions in the Sprite Distributed File System.* Computer Science Division, University of California at Berkeley, Technical Report UCB/CSD 90/567, April 1990.

Appendix C

Thomas, Joseph P. and Salo, Timothy J., *High-Performance Network I/O for a Massively Parallel Processor,* Minnesota Supercomputer Center, Inc., 1992.

This deliverable is referred to as "Summary of Activity Required to Support Additional Supercomputer Architectures" in the VHSRFS proposal.

High-Performance Network I/O for a Massively Parallel Processor

Joseph P. Thomas
Timothy J. Salo
Minnesota Supercomputer Center, Inc. *

December 14, 1992

Abstract

This paper discusses research into issues involved in implementing IP over HIPPI for a massively parallel architecture machine. Using a Thinking Machines Corporation's CM-5 as a model, we propose a design for implementing a HIPPI physical interface with the TCP/IP protocol operating in software. An overview of HIPPI and the CM-5 architecture is also provided.

1 Introduction

The advent of gigabit-per-second (Gb/s) wide-area networks has opened the doors to a new age of supercomputing. For the first time, supercomputers can communicate across wide areas at speeds near that of supercomputer channels. This capability will enable a wide variety of applications which were previously infeasible.

The NSFNET currently supports bandwidth up to 45 megabits per second (Mb/s). While 45 Mb/s is not supercomputer channel speed, design work is underway to provide a backbone capable of delivering 155 Mb/s, with plans to eventually support 622 Mb/s. The Department of Energy and NASA are planning a backbone (ESNET) to support 155 Mb/s with options to upgrade to 622 Mb/s. Clearly the trend of backbone wide-area networks is toward supporting gigabit-per-second bandwidths.

HIPPI, with a data rate of 800 Mb/s, has emerged as the standard inter-machine connection for supercomputers. Already, local-area networks are being created between supercomputers using HIPPI interfaces and HIPPI switches. Today, many computing centers offer a local computing environment in which several different resources may be combined into a single compute system. Such a system might include elements such as:

- A vector supercomputer
- A massively parallel supercomputer
- Very high-speed data storage
- High-resolution graphical displays

Combined, these resources provide a researcher with a very powerful computational tool. With very high-speed wide-area networks in place, one can envision a *network supercomputer* in which the resources of several geographically distributed computing resources can be used as a single computing system, much as today's local environments. For example, such a *network supercomputer* would allow a researcher with a graphical display in one part of the country to manipulate satellite imagery stored at a second remote site using the computer resources of a third site.

Creating such new computing systems will be dependent upon being able to integrate the individual pieces. Given that IP is the prevalent internetwork protocol, and that HIPPI is the de facto standard for supercomputer channels, IP over HIPPI becomes the means of integrating massively parallel machine architectures.

*The research described herein was sponsored by DARPA through the U. S. Army Research Office, Department of the Army, contract number DAAL03-91-C-0049. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by DARPA or the U. S. Army.

1.1 HIPPI

HIPPI (High-Performance Parallel Interface) is a simplex, high-performance, point-to-point parallel interface capable of peak data rates of 800 or 1,600 megabits per second over one or two twisted-pair copper cables. A single HIPPI cable provides a 32-bit wide data path at 800 megabits per second, but two cables may be used to provide a 64-bit data path at speeds up to 1,600 megabits per second. A full duplex connection is created by using two HIPPI physical interfaces, one from source to destination, another from destination to source. Connections need not support the same data rate, ie. A to B may support only 800 Mbit/s while B to A supports 1,600 Mbit/s.

The HIPPI protocol defines a layering with which to provide transport services (figure 1).

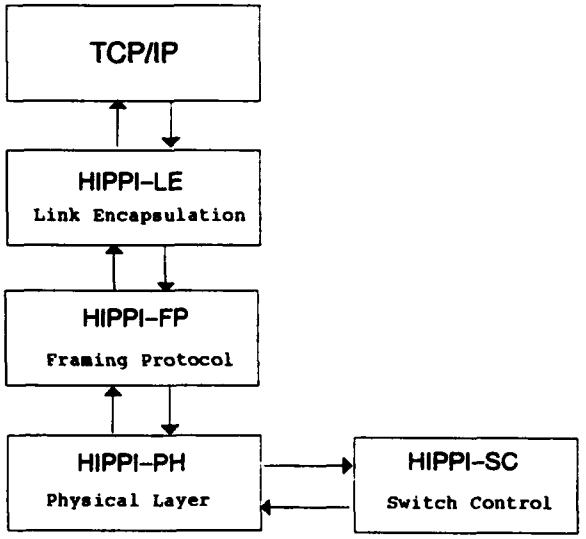

Figure 1: HIPPI Protocol Organization

1.1.1 HIPPI-PH

The HIPPI Physical interface (HIPPI-PH) uses a 50-pair cable providing 32 bits of data, 4 parity bits, and several control lines to implement simple handshaking and a look-ahead flow control mechanism (Figure 2). A HIPPI source indicates its desire to connect to a destination by raising a REQUEST line. After the REQUEST line is raised, the HIPPI Physical Layer (HIPPI-PH) transmits a 32-bit Identification field or I-field. This I-field signals who the source wishes to connect to and may be used as routing information (see section 1.1.4). The destination signals its willingness to accept the connection by raising a CONNECT line. Once connected, the destination will signal its ability to accept data from the source by sending a READY pulse for each "burst" of 256 32-bit words of data the destination can buffer. In this manner, the destination controls the flow of data before the source can begin transmitting.

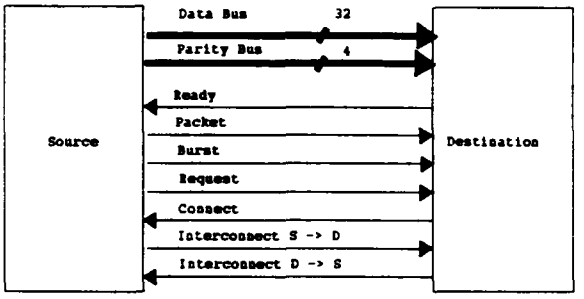

Figure 2: HIPPI Interface Signals

Packets of information may contain up to 2^{32} bytes of data, broken into bursts of 1 to 256 32-bit words of information. One burst is sent from the source to the destination for each READY pulse received. Data is sent with odd byte parity being transmitted over the parity bus with each 32-bit transfer. A length/longitudinal redundancy check (LLRC) is performed for each burst of data and is transferred at the end of the burst (Figure 3).

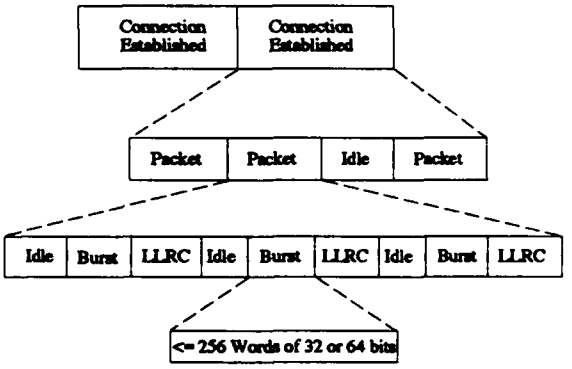

Figure 3: HIPPI Packet Format

1.1.2 HIPPI-FP

HIPPI Framing Protocol (HIPPI-FP) provides a data framing protocol to the HIPPI physical layer (HIPPI-PH). Important characteristics of HIPPI-FP include:

- Separation of user control information and user data information

- Early delivery of user control information
- Support for multiple upper-layer protocols (ULP's)
- Connectionless data service

A HIPPI-FP packet is composed of three areas (figure 4): the **Header_Area**, the **D1_Area**, and the **D2_Area**. The **Header_Area** is 64 bits in length and contains a **ULP-id**, a bit to indicate the presence or absence of the **D1_Area (P)**, a bit to indicate if the **D2_Area** begins on a burst boundary **(B)**, the size of the **D1_Area** and **D2_Area**, and the offset to the start of the **D2_Data_Set**. Note that the size of the **D1_Area** is specified in number of 64-bit words whereas the size of the **D2_Area** is specified in number of bytes.

If present, the **D1_Area** is intended to convey control information. The size of the **D1_Area** is limited to 1016 bytes so that the entire **D1_Area** will fit in the first HIPPI burst. This allows the **D1_Area** to contain control information which may be delivered to the ULP for processing without waiting for additional bursts in the packet to arrive.

The **D2_Area** is intended to convey user data and immediately follows the **D1_Area**. Whereas the size of the **D1_Area** is limited to fit in the first HIPPI burst, the size of the **D2_Area** has no restrictions. **D2_Size** may range from 0, indicating *not present*, to 0xFFFFFFFF, indicating that the size is *unknown*. In the case of the *unknown* size, it is up to the ULP to determine the end of the **D2_Area**.

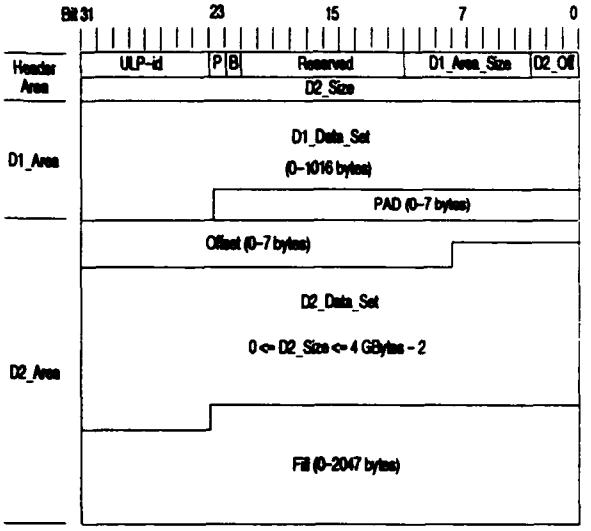

Figure 4: HIPPI-FP Packet Definition

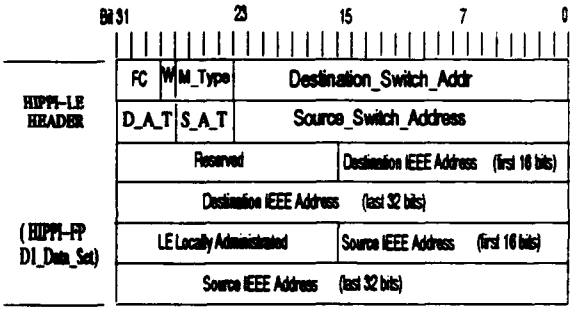

Figure 5: HIPPI-LE Packet Definition

1.1.3 HIPPI-LE

HIPPI Link Encapsulation (HIPPI-LE) performs encapsulation of 802.2 Logical Link Control (LLC) frames (Figure 5) within HIPPI-FP frames. The 802.2 LLC header is contained entirely within the HIPPI-FP **D1_Data_Set** for immediate processing. The HIPPI-LE header contains a 3-bit Forwarding Class (FC), several undefined or reserved fields (*Vendor_Unique*, *Reserved*, and *LE_Locally_Administered*), and two 802.1A IEEE addresses (*Destination_IEEE_Address* and *Source_IEEE_Address*). The 802.2 Protocol Data Unit (PDU) or user data is contained in one contiguous **D2_Data_Set** beginning with the LLC SNAP header (Figure 6). While not currently defined, the FC field could be used to distinguish between types of services, including but not limited to:

- Regular data
- Unreliable data
- Services requiring guaranteed bandwidth
- Network or switch control information

1.1.4 HIPPI-SC

HIPPI Switch Control (HIPPI-SC) is a specification for connecting HIPPI interfaces into a local-area network using HIPPI switches. HIPPI-SC provides support for both source routing and destination address routing protocols.

In order to facilitate local-area networks, HIPPI-SC

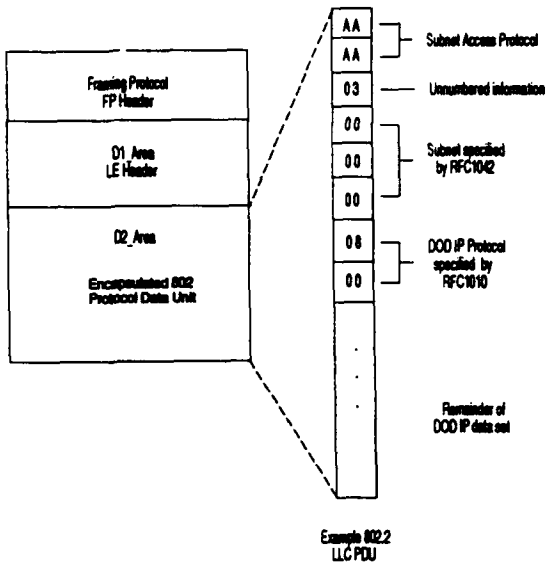

Figure 6: HIPPI-LE SNAP Encapsulation

uses the HIPPI-PH I-field as Connection Control Information (CCI). CCI reserves several bits for use as direction indicators, data-path width, path selection, and camp-on. The remaining 24 bits are used by the switches as the routing control information field.

Source routing allows the routing control field to be divided into subfields which specify which output port of the switch to use. When using source routing, each switch will end-off shift the output port and insert the input port value into the other end of the routing control field. The CCI direction bit indicates in which direction the end-off shift should occur. By changing the direction bit, the destination host has a valid return route to the source host. Source routing requires that the source host know the path to the destination and thus also know the network topology.

Destination address routing splits the routing control field into a source address field and a destination address field. These address fields must be unique within the switch fabric. Destination addressing allows the source host to specify a destination address and allows each switch in the network to make its own routing decision. Destination address routing does not require that the host know anything about the network topology.

Switches can arbitrate fair access when hosts set the CCI camp-on bit while establishing connections. In the case of attempting to connect to a busy host and the CCI camp-on bit is off, the source host will receive a REJECTED indicator. If the source sets the CCI camp-on bit, the switch will wait to complete the connection until the destination is no longer busy. In the case where several hosts are waiting to connect to a single destination, the switch randomly selects which request to complete.

1.1.5 IP OVER HIPPI

RFC 1374, *"IP and ARP on HIPPI"* [33], describes a standard encapsulation method and a scheme for implementing IP and ARP on a HIPPI local-area network.

IP datagrams may be encapsulated into HIPPI packet formats using the HIPPI-FP and HIPPI-LE services. An IP datagram will use a HIPPI-FP packet with the ULP-id set to 4, the **D1_Data_Set_Present** (HIPPI-LE **'P'** bit) will be set with the **D1_Area_Size** set to 3 (the size in 64-bit words of a HIPPI-LE header). The **D2_Area_Size** will contain the number of octets in the 802.2 LLC PDU and will include the length of both the LLC/SNAP header and IP datagram.

IP over HIPPI makes use of the HIPPI-LE **Reserved** fields to indicate message types (Data vs. Address Resolution protocols), switch addresses for routing through HIPPI switches, and the type of switch routing being used. The **HIPPI-LE IEEE_Address** fields contain the 48-bit MAC addresses.

The **D2_Area** will contain the 802.2 LLC/SNAP headers and the IP datagram. The LLC header will contain decimal 170 for the SSAP and DSAP (Subnet Access Protocol) fields and decimal 3 for the Control field. The SNAP header will use an Organization Code of zero, and the Protocol ID as defined in RFC 1060, *Assigned Numbers* [35].

The Address Resolution Protocol (ARP) can be implemented in HIPPI LANs in order to map IP addresses to HIPPI switch addresses. The format of a HIPPI ARP packet will be the same as an Ethernet[1] ARP packet and will be encapsulated into HIPPI in the same manner as other IP datagrams. ARPing on a HIPPI LAN will use either a multicast capability, if available, or a third-party ARP agent. In a multicast environment, the target node will fill in the missing information from the ARP request and send a reply

[1] Ethernet is a registered trademark of Xerox Corporation.

to the ARP requestor. In a LAN which does not support multicast, the requestor will connect to a known third-party agent who will then reply as if the target node had received the ARP request.

1.2 CM-5 ARCHITECTURE

The CM-5 is a scalable high-performance parallel processor architecture designed to be capable of delivering performance into the teraflop range while providing support for both the SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data) parallel programming models. To achieve this goal, the CM-5 consists of several communication networks which tie together a series of general purpose processors or *nodes*. Each node is built around industry standard microprocessor technology, allowing the nodes to be replaced with higher performance processors as they become available.

1.2.1 NODES

Nodes on the CM-5 are broken into two categories. The first category is the computing node, also known as a Processing Node or PN. The second category, Control Processor or CP, covers all other types of nodes.

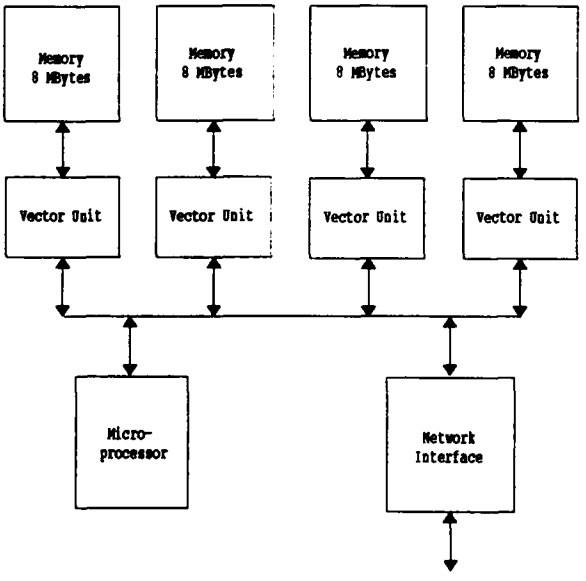

Figure 7: Vector Node Processor

Processing nodes are built around SPARC RISC processors and include four vector units, a total of 32 megabytes of local memory, and a network interface (NI) port (Figure 7). Being a general purpose microcomputer, each processing node is able to act independently in MIMD mode, or in conjunction with other nodes in SIMD mode. In MIMD mode, different nodes carry out different instructions on separate data elements stored in their local memory. Having one processor handle the 'WHERE' clause of a loop while another processor handles the 'ELSEWHERE' clause is a MIMD function. In SIMD mode, each node in a group of nodes carries out the same instruction on the same data element in its own local memory at the same time. Executing the instruction C = A + B where A, B and C are large arrays can be carried out in SIMD mode where each processor 'n' carries out the instruction C[n] = A[n] + B[n].

Vector units are specialized hardware arithmetic accelerators. Each vector unit has its own eight-megabyte section of memory and, in addition to performing arithmetic functions, acts as a memory controller for the microprocessor. Each vector unit contains 64 64-bit registers which also may be treated as 128 32-bit registers. Receiving instructions from the microprocessor, the vector units may act independently or be grouped together, allowing operations on data up to 256 bits. This provides a maximum performance of 128 MFlops (Millions of Floating Point Operations Per Second) per processing node.

Control nodes perform the non-parallel processing tasks for the CM-5. Many control nodes are general purpose UNIX workstations which provide such functions as user support, time-sharing, accounting, diagnostics, and I/O control. Such nodes often include additional I/O services such as local disks and connections to wide area networks.

Control nodes also may be dedicated to specific functions. Examples of specific functions include partition managers (PMs) and I/O control processors (IOCPs).

Partition managers group the processing nodes into one or more virtual machines called partitions, each controlled by a partition manager (Figure 8). Each partition operates independently from other partitions and cannot affect other parts of the machine.

I/O control processors are designed to perform I/O services on behalf of the processing nodes. IOCPs attach to the CM-5 via network interface ports in the same manner as processing nodes. All access to external I/O, such as data storage and network access,

occurs via the IOCP. An important feature of the I/O architecture is the ability of the IOCP to aggregate I/O and thus be configured to provide any desired bandwidth into and out of partitions.

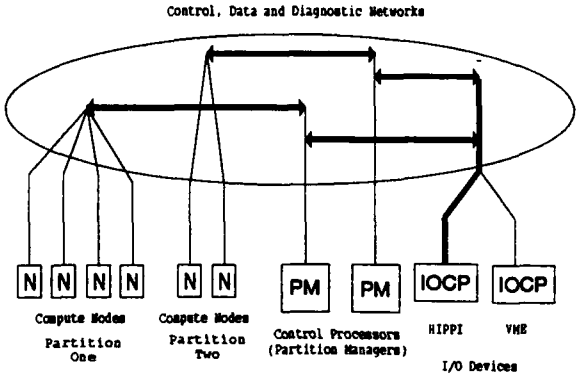

Figure 8: CM-5 Organization

1.2.2 NETWORKS

Every node in the CM-5 is tied to every other node via a series of networks. The CM-5 contains three networks: the Control Network, the Data Network, and the Diagnostic Network, each designed to perform a specific task. Each of these networks is scalable independent of the type of node hardware and can conceivably address over one million nodes.

The Control Network is responsible for controlling the interoperation of the processing nodes and can be thought of as the SIMD part of the CM-5. The Control Network handles those functions which involve all of the processing nodes. Control operations include broadcasting, combining, global logical operations, and synchronization.

Broadcasting is used to send data to all of the nodes within a partition. The partition manager uses broadcasting when the user program is sent to the processing nodes. Processing nodes may use broadcasting when a single node wishes to distribute a single value to all other processing nodes.

Combining can be used to do data reduction or parallel prefix operations. The combine unit handles five operations in the hardware of the Control Network. These five are bitwise logical OR, bitwise logical XOR, signed integer maximum, signed integer addition, and unsigned integer addition. An example of data reduction would have each processing node place a value into the combine unit, and when all processing nodes have sent their data, each node would receive the sum of all values from all nodes. A parallel prefix operation could have each node place a value into the combine unit and when all nodes have finished, each node would receive the maximum value of all values from all nodes.

Global operations produce the logical OR of one bit from all processing nodes. This logical OR is used as a signaling protocol for various conditions and exceptions. Global operations can be done via one synchronous or two asynchronous units and are independent of each other and other Control Network activity.

The synchronization function combines the global unit and the combine unit to perform a barrier synchronization function. Each processing node writes a message to the combine unit and waits for the global unit to signal that every node has received the result. By waiting for the **global_done**, no processor will proceed past the barrier until every processing node has reached the barrier.

The Data Network is responsible for the bulk transport of data between nodes and implements the MIMD side of the CM-5. Each processor can simultaneously send messages to be delivered in a point-to-point fashion as long as there is room within the Data Network. Internally, the Data Network is divided into both a Left Data Network and a Right Data Network. Users can either specify which half to use or allow the CM-5 to send on the most empty half. An important function of the Data Network is that it guarantees delivery of the data and does not rely on the processing nodes for end-to-end delivery. From a processing node perspective, each message sent is guaranteed to arrive at its destination.

Each processing node is connected into the Data Network via a Network Interface (NI) port. The Data Network uses a variation on a binary fat-tree interconnect to provide for high bandwidth communications. Each node has two connections to the first tree level, providing for a raw bandwidth of 40 megabytes per second in and out. Each of the first two levels of the Data Network also use two connections, yielding a 16 node grouping with an aggregate bandwidth of 160 megabytes per second. Above the second level, four connections are used for all remaining levels, providing a linearly scalable bandwidth.

If we define a group of 2^k nodes, we can say that any two nodes are within that group if their ad-

dresses differ only in the lowest k bits. This grouping has some significant features for I/O throughput. Any two nodes within a group of four can sustain 20 Mbytes/sec. of data. Any two nodes within a group of 16 can sustain 10 Mbytes/sec. of data. Any two nodes outside of a group of 16 can sustain 5 Mbytes/sec. regardless of the size of the group since we are now beyond level two in the network tree. This implies that the best to worst case of bandwidth is only a factor of four and is very predictable. By using the grouping and scalability, we can easily achieve several gigabytes of aggregate bandwidth. For example, within a group of four nodes, the aggregate bandwidth is 80 Mbytes/sec. (4 × 20 Mbytes/sec.). Within a group of 16 nodes, the aggregate bandwidth is 160 Mbytes/sec. (16 × 10 Mbytes/sec.). Within a group of 64 nodes, the aggregate bandwidth is 320 Mbytes/sec. (64 × 5 Mbytes/sec.) and within 512 nodes, the aggregate bandwidth is 2.56 Gigabytes (512 × 5 Mbytes/sec.).

The third network is the Diagnostic Network and is not available to the general user. The Diagnostic Network was designed to connect to each part of the CM-5 and allow parallel testing of all the nodes in the machine and to provide the ability to map faulty parts out of the system. This control allows the engineers to isolate faulty systems and to continue to provide users with a logically complete system. That is, any group of nodes can be removed from the system and another group logically mapped to occupy its addresses. This design allows for a very robust machine with minimal impact on users.

2 REQUIREMENTS

There are several requirements for efficiently implementing a high-speed TCP/IP interface. This section outlines several requirements which a high-speed interface should achieve. A high-speed network interface should:

- Support standard channel technology
- Operate at supercomputer channel speeds (600-800 Mb/s)
- Provide separation of IP services from user services
- Support extensions to TCP
- Provide expandability

Support standard channel technology

The key to being successful in today's networks is *interoperability*. Today's supercomputers are typically integrated into local area networks using HIPPI technology. Massively parallel architectures will need to co-exist with existing LANs and thus must support the HIPPI protocols.

Operate at supercomputer channel speeds

While perhaps apparent, it needs stating that a network interface should be capable of sustaining the needs of the computer. If the computer is capable of delivering hundreds of megabytes of data per second, it should not be fed or drained with a 10 megabit channel. Using HIPPI as a network interface, with data rates of 800 Mb/s or 1,600 Mb/s, is a good choice.

Separation of services

With massively parallel architectures, there is a question of who should provide the IP protocol services. An application running on a parallel architecture has expectations that it will be evenly distributed among available processors. Many parallel algorithms expect that the number of processors available will be a power of 2. In addition, the process of performing TCP/IP protocol processing across N arbitrary processors is not readily understood. For these reasons, application processors probably should not perform protocol processing. Rather, protocol processing should be handled by one or more external processors separate from the application processors.

Support TCP extensions

Existing TCP protocols have design limitations which restrict their ability to utilize high bandwidth channels. Several extensions to TCP have been proposed which, while maintaining compatability with existing protocols, allow high bandwidth channels to be more fully utilized. RFC 1323, *"TCP Extensions for High Performance"* [7], describes several extensions to TCP such as a window scale option, a method of measuring round-trip times, and a means to protect against wrapped sequence numbers.

Provide expandability

With many of the new parallel architectures being scalable, it follows that the network services also must be able to scale. While we currently talk about wide-

area networks with speeds of 155 or 622 Mb/s, these speeds will eventually be surpassed and will begin to approach 64-bit HIPPI speeds at 1,600 Mb/s. This implies that network interfaces also should be able to scale and support 1,600 Mb/s HIPPI.

3 PROPOSED DESIGN

This section discusses some of the issues driving a proposed design. We look at how we can achieve the stated requirements by separately examining the hardware and software issues. At the conclusion, we recommend that an IOCP be dedicated to handling both the HIPPI and IP protocols.

3.1 HARDWARE INTERFACE

Examining how supercomputers are used today, we discover that the average user is no longer physically close to the machine but is more often geographically distant and communicates over wide-area networks. Trends are emerging which demonstrate that wide-area networks are moving toward supporting high-speed bandwidths on the order of 155 Mb/s or 622 Mb/s.

Examining existing channel technology, we find that the predominant supercomputer channel is HIPPI based. Supercomputers today are being interconnected locally via HIPPI switches creating high speed local-area networks.

If we are to choose a technology standard capable of interfacing to these local-area and wide-area networks, HIPPI is the most widely used technology available. Capable of supporting either 800 or 1,600 Mb/s, HIPPI can deliver the bandwidth for today, tomorrow, and beyond.

The design of a HIPPI interface intended to support TCP/IP is governed by several items:

- **HIPPI I/O Speed** - Given that HIPPI operates at either 800 Mb/s or 1,600 Mb/s, how will the interface connect into the parallel architecture?
- **HIPPI and TCP are Byte Stream Protocols** - Since both HIPPI and TCP transfer data as an ordered stream of bytes, how does data from individual processing nodes get properly ordered?
- **Protocol Overhead** - Who is responsible for generating headers, checksums, and making routing decissions for the IP protocols?
- **Data Buffering** - In a timeshared operating system, who manages data until the user process is ready to read it?

We briefly examine three models (figures 9–11) for providing TCP/IP. Model one, currently used by the CM-5, uses a proprietary (CMIO) channel to move data into and out of the machine, an external device (Sun-4) to convert from the proprietary channel to a standard channel, and an external protocol processor. Model two would use standard channel technology such as HIPPI in and out of the CM-5, but still make use of an external protocol processor. Model three would again use HIPPI, but would move the protocol processing onto the channel processor.

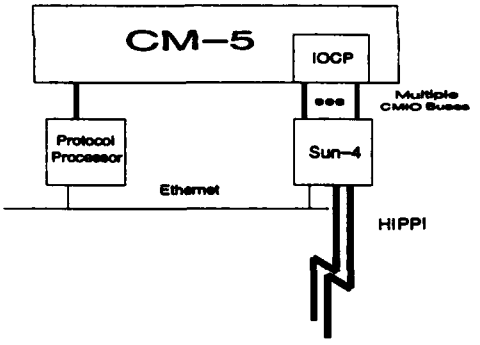

Figure 9: Model One - Current CM-5

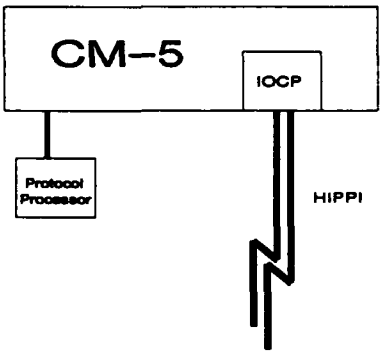

Figure 10: Model Two - Standard channel interface

Model one (figure 9) has several drawbacks including:

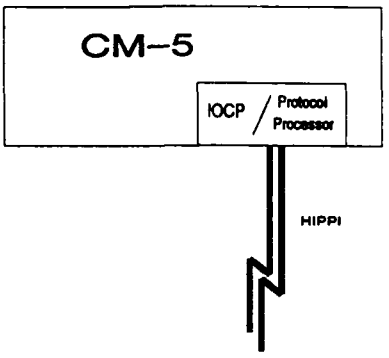

Figure 11: Model Three - Internal protocol processing

- The CMIO bus is slower than HIPPI, requiring aggregating multiple CMIO channels into a single HIPPI channel
- Requirement for protocol conversion from CMIO signalling to HIPPI signalling
- Use of hardware external to the machine to perform the CMIO to HIPPI conversion

Model two (figure 10) moves to a standard channel interface, i.e. HIPPI, but retains an external protocol processor. Drawbacks of model two are its separation of data and control processing, and the amount of external communication needed between the network interface and the protocol processing engine.

The perferred model, model three (figure 11), moves the protocol processor from model two into the network interface. Such a model allows data and control information to remain together and does not require external communication.

Massively parallel architectures can introduce additional complexity by requiring data ordering. On machines such as the CM-5, where processing nodes are complete machines in themselves, data ordering can be achieved by ordering reads or writes to the collection of nodes. Such machines are byte-oriented allowing direct mapping to the network I/O stream. On machines such as the CM-200, data is bit-oriented. Additionally, what is considered neighboring data may not reside in neighboring processors. On the CM-200, data must first be arranged from bits into bytes, and then from bytes into a network ordered I/O stream. Thus, the difficulty of data ordering differs from machine to machine.

IP protocol handling should be handled either by the IOCP or another similarly dedicated processor. This protocol engine will handle all the traditional kernel level support most UNIX[2] operating systems provide. This includes buffering of data between user processes and the network, HIPPI LLRC or TCP checksum generation, headers, routing, and other control messages. The protocol engine should be capable of receiving and sending messages independent of user requests. A large amount of buffering should be provided in order to hold packets until a user process is ready to read them. Allowing for large TCP packets (64K bytes), large TCP windows for large *Bandwidth* $\times$ *Delay* networks, and (re)transmission buffers, several megabytes of buffer memory would be needed.

Because TCP/IP does not require a user action to stimulate a response, there are data buffering problems to be dealt with. The case of a user attempting to transmit data is a relatively simple task. The protocol engine waits until the users signals that data is available. When the user is ready, data conceviably can be transmitted directly from user space to the network media assuming that protocol headers are built and sent in the correct order.

It is not always the case that data may be sent and discarded immediately. At times, the destination the user wishes to converse with may not be ready to accept HIPPI connections. When this happens, the protocol engine must either block the user process or buffer the data for the user until a connection can be established and the data transmitted. In some cases, unreliable or congested networks may cause data in transit to be lost. It may not be desirable to block the user process when external buffering and processing can handle the connection delay or reliable delivery of data.

The receive side becomes more complicated when we begin to consider several scenarios. In an ideal case, the user process desiring the data is ready and waiting and, as data arrives, the protocol engine may transfer it directly into the user's application. This is pretty rare. The more common case is that the user process requesting the data is either not in memory (i.e. the operating system has given this time slice to some other user) or, if in memory, the user process has not yet posted a *read* request for the data.

The third requirement is the ability to move the protocol processing away from the user. Protocol processing includes generating headers, computing and filling checksum fields, and making routing decisions

[2]UNIX is a registered trademark of UNIX Systems Labratories, Inc.

based on destination address. There is also "hidden" processing in the form of address resolution, dynamic routing protocols, and network control messages. While it is certainly possible for the user to deal with such issues, the accepted model is to have some agent perform these on behalf of all users.

As this discusion indicates, an IP interface requires that there be some method of buffering and processing data independent of the user or the application.

We recommend the use of an IOCP to implement the IP protocol suite and to manage the data transfer with the user partition. The IOCP provides the protocol engine and data ordering operations necessary to move data between processing nodes and the HIPPI interface and separates the IP protocol processing from the user. Requests are made from a user partition to the IOCP signaling a data transfer to occur. The IOCP must determine the data distribution among processing nodes and order a data transfer to occur such that the overall byte-stream is maintained. The IOCP will handle the low-level HIPPI protocol including establishment of connections and asserting burst READY signals.

It is expected that there will be HIPPI traffic without a user process. Therefore, the IOCP should be able to generate and receive packets without an outstanding user request. It will be up to the IOCP to decide if a packet is destined for the IOCP or for a user process by examining the **ULP-id** field of the HIPPI-FP header and the encapsulated IP header. Control packets, or other information packets for the IOCP, will have unique ULP's. The decision to retain or abort other packets will be dependent upon information received from the user process.

3.2 SOFTWARE INTERFACE

The software side of the implementation has two aspects—that of the user interface and that of handling the TCP/IP protocol. We examine several issues with existing TCP/IP protocol implementations which have an adverse effect on network performance.

3.2.1 USER INTERFACE

On the user interface front, the de facto standard has become the Berkeley Socket Interface based on the BSD 4.3 networking release. The overall goal is for the user to take software written to the BSD networking specs and run it without modification on the CM-5. Users create sockets into the network layer and then send and receive data using standard system calls. Socket types of STREAM (connection oriented), DGRAM (connectionless) and RAW (internal network access) should be supported. Standard Berkeley socket calls such as *connect*, *bind*, *listen*, *accept*, *send* and *recv* should all be supported.

3.2.2 PROTOCOL DIVISION

The protocol engine portion of the task will be, by far, the hardest to complete properly. Critical issues for the engine include how to support the multiple partitions and how to sustain the 100 Mbyte/sec. data rate. In addition, with the emerging interest in gigabit networks, the CM-5 should be readily adaptable to changes in protocol in order to participate in various network testbeds.

One of the first issues is how to configure the CM-5 in terms of its appearance as a host to the network. The ability to reconfigure the number and size of partitions in effect makes the machine appear as multiple hosts. Whereas each host or partition needs to be assigned a unique hostname and IP address, providing a unique HIPPI interface for each partition becomes an expensive proposition. On the reverse side, limiting the entire CM-5 to a single HIPPI interface for IP will present a tremendous bottleneck as the number of processing nodes grows into the thousands, with tens of partitions. In order to provide flexibility, the protocol engine needs to be capable of supporting some number of partitions over some number of HIPPI interfaces. As the number of HIPPI interfaces grows, it may become desirable or necessary to add additional protocol engines. In this case, communication between the individual protocol engines should be as TCP/IP peers and not via proprietary protocols. Figure 12 shows several partitions sharing a single protocol engine with multiple HIPPI interfaces, while another partition has a separate protocol engine and HIPPI interface. The protocol engines should be capable of communicating both internally and externally (via a HIPPI network) using the TCP/IP protocols.

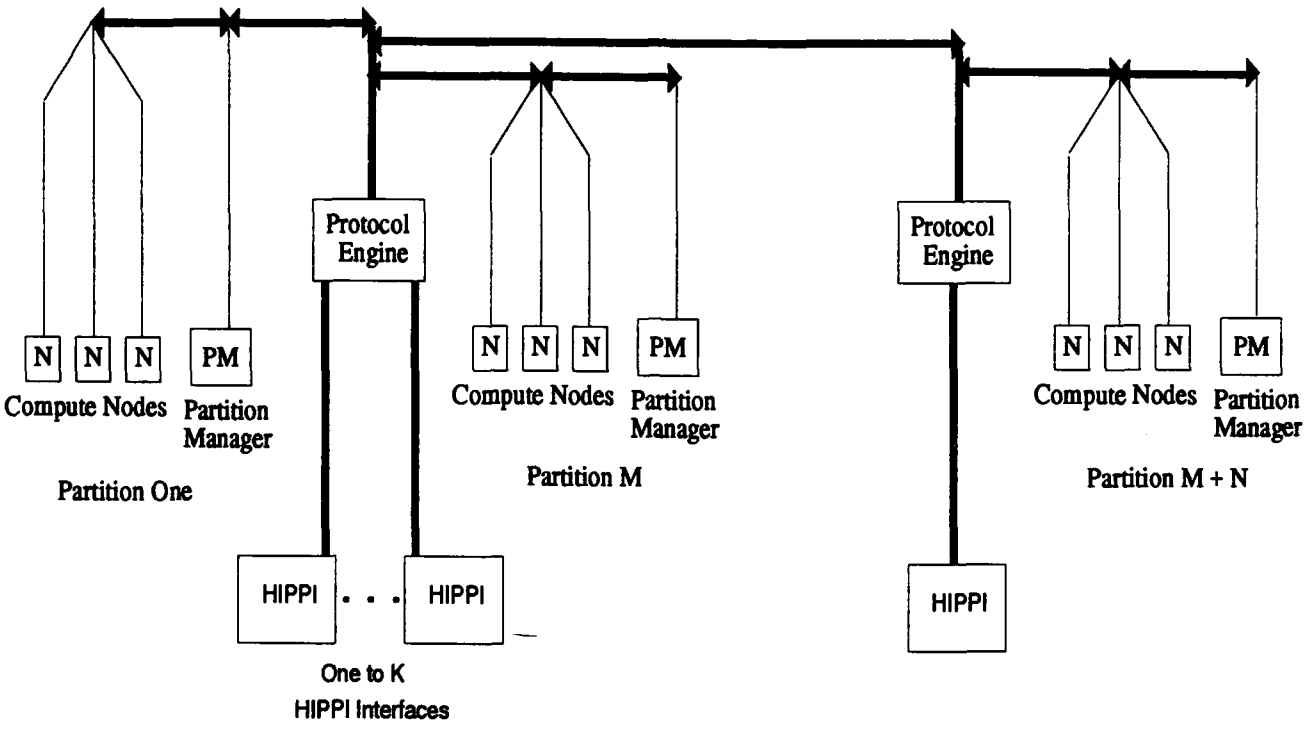

Figure 12: Multiple Protocol Engines and HIPPI Interfaces

3.2.3 PROTOCOL LIMITATIONS

Already, research into higher speed networks has identified that there are limitations with current TCP/IP implementations. Whereas many people would claim that these limitations prevent TCP from becoming a gigabit transport protocol, many studies have shown that this is not the case ([2], [3], [6], [7], [14], [24], [25], [26], and [27]). By carefully examining where the limitations are, solutions can be found which do not violate interoperability with existing TCP/IP implementations. To support this claim, Cray Research Inc. has measured TCP throughput using the software loopback at 795 Mbits/sec. [3]. Cray Research Inc. claims that with additional work, the TCP data rate could easily exceed 1 gigabit/sec.

3.2.4 WINDOW SIZE

TCP's ability to utilize a channel can be characterized by the channel's *Bandwidth* × *Delay* product. We begin by defining

$$\mathbf{B} = \textit{Bandwidth} \times \textit{Delay}$$

where *Delay* is the propagation delay of the channel. As **B** becomes large, due either to very high bandwidths (such as HIPPI) or very large delays (such as satellite), noticeable drops in throughput are encountered. When this happens, the TCP window size is limiting the amount of unacknowledged data which can be present in the channel.

The TCP protocol defines a window of data which can be transferred before some acknowledgement of receipt is required. Existing protocol definitions limit this size to a 16-bit integer, allowing for a maximum window size of 64 Kbytes. At full speed, HIPPI can transfer almost 1,600 times this amount of data per second. Assuming full speed HIPPI and data traveling at the speed of light, a HIPPI link can span no more than 60 miles before the sender will be required to stop and wait for acknowledgements before continuing with the next burst of data. Given that data will not travel at the speed of light, and that LANs are likely to be composed of one or more switches or gateways, each introducing some delay, it is conceivable that even very small LANs will see noticeable gaps when transmitting at HIPPI data rates. A solution to this problem is the window scaling option described in Appendix A.

3.2.5 SELECTIVE ACKS

As the window size increases, so does the amount of data in transit. With the current positive acknowledgement scheme, the sender must buffer all data until an acknowledgement is received. Only then can the sender discard that data and move his window forward. If we look at a case where only one packet out of several thousand is lost, and that packet happens to be near the front of the sending window, TCP will be asked to hold and potentially retransmit several megabytes of data. One solution is to use Selective ACKs (see Appendix B) which provide a more detailed picture of data received. Thus, the sender, upon receipt of a SACK, can discard all data which was correctly received and reduce the buffering and retransmission overhead.

RFC 1106 notes that SACK's require a large amount of state information to be maintained, and that an implementation's desire to delay sending ACKs or SACKs to reduce chatter and increase information content can in itself cause delays (Appendix C). They proposed an additional feature called a negative acknowledgement or NACK. The NACK can be used immediately upon TCP noticing a missing segment, and need not be delayed until some large number of packets has been received. Since a large number of consecutive lost packets could produce a large number of NACKs, both SACK and NACK can be used together to reduce packet processing of NACKs and to improve performance over ACKs alone.

3.2.6 ROUND TRIP TIME

Data retransmission is dependent not only on receiving information from the recipient, but also on the lack of information. That is, if no response is received within some time period, the sender assumes that the data never arrived and retransmits it all. The time period within which to expect a reply is based on a sampling of the round trip time (RTT). This estimate is usually based on one sample taken over a large number of packets within a segment. If we use larger window sizes to gain better performance, or when dealing with lossy transmission lines, these estimates become more and more unreliable. In order to make better estimates, an option for timestamps was introduced (Appendix D). This estimate allows for more accurate RTT's, reducing unnecessary retransmissions and helps solve yet another potential problem.

3.2.7 SEGMENT LIFETIME

As the *Bandwidth* $\times$ *Delay* product is increased, yet another TCP limitation is encountered, that of sequence numbers. Sequence numbers are used to protect against receiving duplicate packets or packets which exist in the network from a previous session. Sequence numbers can be wrapped in as little as 17 secs. at gigabit speeds. This value is arrived at by the need to satisfy the condition $2^{31} \div B > MSL$, where B is the bandwidth in bytes per second, and MSL is the Maximum Segment Lifetime. RFC 1232 solves for $Twrap = 2^{31} \div B$ and presents table 1 [7]. With a maximum segment lifetime of two minutes, this allows for the possibility of several different packets with identical sequence numbers to be received during a number's life span. Some method must be imposed to distingusih packets which are valid from packets that are either duplicates or the result of previous sessions. RFC 1323 proposes an algorithm, using the TCP Time-Stamp option, which protects against wrapped sequence numbers (Appendix E).

Network	Bits/sec. B $\times$ 8	Twrap (seconds)
ARPANET	56kbps	3×10^5 (~3.6 days)
DS1	1.5Mbps	10^4 (~3 hours)
Ethernet	10Mbps	1,700 (~30 minutess)
DS3	45Mbps	380
FDDI	100Mbps	170
Gigabit	1Gbps	17

Table 1: Sequence Number Wrap Times

3.2.8 HEADER PREDICTION

TCP performance is by no means limited only to limitations in the protocols. Several studies have shown that current TCP protocols are capable of sustaining several hundred megabits of data transfer if the overhead processing has been optimized [2, 3, 24, 25, 27]. By far the largest consumer of packet overhead is the examination of headers and the computation of checksums. One of the more recent approaches to header processing is that of header prediction [20]. Header prediction assumes that a packet is being received in order and is correct. In this case, TCP performs a minimum number of validity checks before passing the packet on. Only when one of the checks fails is a more complete or traditional examination performed.

3.2.9 PATH MTU

With the widespread use of higher speed networks, the path maximum transmission unit (MTU) has also become an issue. Current IP calls for using MTU sizes of 576 bytes for non-local networks. Over HIPPI, this will cause an enormous amount of small packets and defeat many of the performance enhancements. Instead, IP should use a path MTU discovery, as described in RFC 1191 [15], to use the largest possible MTU size (Appendix F). Path discovery starts with the largest possible MTU size, and reduces the MTU only when a part of the path is unable to forward the datagram. As HIPPI devices most likely talk to other HIPPI devices, this provides a great improvement over the default 576 byte MTU.

3.3 EXTENSIONS

Massively parallel architectures present some unique challenges for protocol processing software. Consider the case of checksums, where the original data is spread amongst multiple processors. Because the checksum function is associative, it is possible to have each processor generate a partial checksum of data it alone holds. As the partial sums are available, they may be summed to generate the protocol checksum. By reducing the length of data over which the checksum is being computed, the time required to finish the computation is reduced. On the CM-5, this becomes an especially interesting case as the control network is capable of generating sums in hardware. If each node were to compute the local checksum and send that value to the control network, the combined sum could be computed in hardware and sent to the protocol engine.

One can carry the issue of checksums a step farther when using HIPPI as a transport medium. The placement of checksums has always provided issues for lengthy debate. Current TCP protocols place the checksum in the TCP header ahead of the data. This requires that the checksum be calculated over the data before the interface may begin transmitting the datagram. In the case of HIPPI as a media, we wish to transmit very large amounts of data to make better use of available bandwidth. An area which now bears further study is whether TCP should optionally allow the checksum to be placed at the trailing end of the data. This would allow the interface to begin packet transmission immediately, and allow hardware to compute and insert the checksum as the data is passed out the interface.

Other topics for further study include issues related to HIPPI and how the network is designed. RFC 1374, *"IP and ARP Over HIPPI"* [33], calls for restrictions on the number of bursts which may be sent before dropping the HIPPI connection. This draft assumes that the environment in which HIPPI is being used is a local-area network based on HIPPI switches. The limit on the length of connections was intended to introduce a fair-access protocol when several hosts were attempting to connect to the same destination. In cases where a single host is connected to a single wide-area gateway, it may be appropriate to disregard the duration limit in favor of reducing or eliminating connection-setup overhead.

One problem often seen when connecting to high-speed networks is the per-packet overhead. Many of today's processors receive an interrupt for each and every packet received. For every interrupt received, the processor must save its current state and begin a new subroutine to process the incoming datagram. Once datagram processing is complete, the processor must restore its previous state. The overhead required to switch states can be quite high and often limits the number of packets which can be processed in this manner. Furthermore, looking at wide-area traffic characteristics, studies have shown that more than 84% of TCP traffic is below 256 bytes in length and more than 97% of UDP packets are less than 150 bytes in length [36][37]. These characteristics lead to a large number of very small packets which causes very inefficient use of network bandwidth.

One solution to both the interrupt problem and the small packet problem would be to combine datagrams destined for the same host. Such a combination would require changes to the use of the HIPPI-FP service. We propose a new **ULP-id** which would indicate that multiple HIPPI-FP frames are present. The receiver would treat any data received between HIPPI PACKET signals as a stream of one or more HIPPI-FP frames. Each individual HIPPI frame would contain a single IP datagram, encapsulated as recommended in the draft standard using HIPPI-LE framing. The issue of determining end-of-frame when the **D2_Size** is *unknown* can be handled by requiring that such a frame be the last HIPPI-FP frame in a HIPPI packet.

Gigabit networking is only now becoming a reality. While much research has been done on using IP at gigabit speeds, much more work has yet to be started.

As networks continue to grow, both in size and speed, new problems and limitations will continue to be discovered.

3.4 RECOMMENDATIONS

We believe that supercomputers will continue to be integrated into local area networks rather than directly attached to wide area networks. Thus massively parallel machines will need to support HIPPI as a channel interface. Protocol processing should be handled by separate protocol engines and not shared with application processors. Where appropriate, specialized hardware should be employed to assist processing, but not where it limits protocol change. Hardware assisted checksum or other error-checking codes are desirable cases, whereas generating protocol headers is not. It is desired that only the HIPPI physical interface be visible externally and that protocol engine hardware be kept within the confines of the massively parallel machine.

For the CM-5 this translates into an IOCP which interfaces to the processing nodes via the control and data networks, and provides a HIPPI physical interface externally. The IOCP will provide the protocol engine and memory for data buffering. Control network hardware can be utilized to assist in checksum calculations.

The software implementation of TCP/IP on a massively parallel architecture should observe the statement made in the Host Requirements document (RFC 1123 [29]):

> A vendor who develops computer communication software for the Internet protocol suite (or any other protocol suite!) and then fails to maintain and update that software for changing specifications is going to leave a trail of unhappy customers.

First and foremost: follow the specifications. Secondly:

- Adhere to the Host Requirement documents, RFC 1122 and RFC 1123 [28, 29]
- Follow RFC 1374, *"IP and ARP on HIPPI"* [33]
- Implement TCP extensions for Window Scaling, Time Stamps, PAWS, and IP Path MTU Discovery [7]
- The use of SACKs and NACKs should be further investigated and, if appropriate, implemented in the TCP protocol.
- Participate in the IETF

That being said, the most important recommendation is to be flexible. Gigabit networking is a new technology and many of the standards are still evolving. Realize that needs or specifications will change and be ready to adapt.

Appendicies

A Window Scaling

Traditional TCP uses a 16-bit integer to declare the window size, thus limiting the maximum amount of data in transit to 64 Kbytes. While this may seem to be a large amount, at gigabit speeds, the data pipe is capable of accommodating much more. To overcome this limitation, RFC 1323 [7] redefines the window scale to be a 32-bit integer, and then uses a scale factor to carry this value in a 16-bit integer field.

Included in a SYN segment is an offer to do window scaling and a shift count. If both sides agree to do scaling, then the sender will shift his window size to the right by shift_count bits, and the receive will shift the received window size to the left by shift_count bits. Note that the window size becomes logarithmic by powers of two and that a shift count of zero indicates a window scale of 1.

The left edge of the sender's window must be within 2^{31} bytes of the right edge of the receiver's window. This allows TCP to determine if a segment is "new" or "old" by testing if the sequence number is within 2^{31} bytes. Placing this limitation on the window allows for a maximum scale factor of 14 permitting a 1 gigabyte window.

The scale factor applies only to the window field as transmitted in the TCP header and does not apply to such values as the "congestion window" computed by Slow Start of Congestion Avoidance.

B Selective ACK

Window acknowledgements are designed to give positive response that data has been received. Most TCP implementations try to acknowledge as much data as possible, thus delaying sending ACK packets. In addition, in many cases, only a single packet is lost, but with ACK, the sender may retransmit all data from the lost packet through the end of the window, possibly leading to enormous amounts of unnecessary retransmits.

RFC 1072 [6] describes a Selective ACK (SACK) which can tell the sender about all segments which were received correctly. While not allowing the window to advance beyond the last ACK'd data point, it does allow the sender to retransmit smaller amounts of data. In addition, SACK can be used to convey information about multiple gaps, thus giving an exact picture of which segments were received and which ones remain missing.

C Negative ACK

Negative ACKs (NACK) are described in RFC 1106 [5] as a means to provide immediate notification to the sender that a segment has been lost. Traditional TCP processing tends to delay sending of ACK packets in favor of conveying more information in a smaller number of packets. In lossy networks the NACK can be sent immediately upon loss detection and may allow for the dropped segment to be retransmitted with less waste of bandwidth.

One implementation shown in RFC 1106 shows that, with a 10^{-6} error rate, the use of NACKs can improve throughput by two to four times over that using ACKs alone.

D TimeStamps

Estimates for the Round-Trip Time (RTT) are used to adapt to changing traffic conditions and to retransmit unacknowledged data. If estimates are made at only one sample per window, aliasing is introduced (and the more in error the RTT is). In addition Zhang [32], Jain [30], and Karn [31] have shown that, if retransmitted segments are included, it is impossible to make reliable RTT estimates.

RFC 1323 [7] introduces a TCP Timestamp Option which would allow every segment to carry a timestamp value. Once negotiated, the current timestamp value of the sender is included with every packet. In addition, ACK packets will carry the timestamp value received from the remote end. RFC 1323 considers three cases, that of a delayed ACK, a lost segment, and a filled hole, and presents an algorithm for covering all three cases and what inferences can be made for each situation.

E PAWS

Protect Against Wrapped Sequence numbers (PAWS) is used to reject old or duplicate segments. In very high *Bandwidth* $\times$ *Delay* networks, sequence numbers may wrap long before the Maximum Time to Live (MTL) has expired, thus allowing for different packets to contain duplicate sequence numbers simultaneously. PAWS uses the same Timestamp option described above with the assumption that every received TCP segment will carry a monotonically nondecreasing timestamp. This allows the assertion that a segment can be discarded as old if the timestamp is less than some timestamp already received in this connection.

PAWS works by initializing the starting timestamp to the value received from the SYN phase. Once initialized, if a packet arrives with timestamp less than the last valid timestamp, then the segment is unacceptable and we can send an acknowledgement and drop it, as per RFC 793. If the segment is outside of the window, the segment is rejected, as per normal TCP processing. If the segment is within the window, record its timestamp as the last valid. If the packet is the next in sequence, accept it normally, otherwise queue it as out-of-sequence for later delivery to the user.

RFC 1323 goes on to discuss some issues about the timestamp clocks such as not being too slow, i.e., it must tick at least once for each window, and not being too fast, i.e., it must not wrap within MSL seconds. Furthermore, a discussion of how to handle outdated timestamps and RST segments is presented.

F Path MTU Discovery

Path MTU discovery is a means by which unnecessary IP fragmentation can be avoided. Currently, IP bases its MTU size on the destination address. If the destination is determined to be a directly attached-network, then the MTU will be set according to the MTU of the network media. MTU's for various media are 1500 bytes for Ethernet, 4352 for FDDI and 65,535 for Network Systems Corporation Hyperchannel or HIPPI. For a non-connected network, the current practice calls for selecting the minimum of 576 or the MTU of the first hop. While such a choice allowed early IP to operate across X.25 networks or ARPANET, most of today's networks no longer impose such a small MTU. In the case of using HIPPI or ATM, such a small MTU will have a disastrous effect on throughput.

RFC 1191 introduces Path MTU Discovery. This technique uses the Don't Fragment (DF) option of IP. The host begins by sending all packets with a MTU equal to its interface MTU and setting the DF bit in the IP header. As the packets travel through the network, intervening gateways will either pass the packet unchanged, or discard the packet and return an error. Upon discarding a packet, the gateway will generate an Internet Control Message Protocol (ICMP) "Destination Unreachable" message with an error code of "fragmentation needed and DF set". Upon receipt of such an ICMP message, the sending host can then reduce the MTU and retransmit the packet until the MTU is to a point where the packets reach the destination.

While there are further limitations explained in RFC 1191, use of the Path MTU Discovery option can reduce protocol overhead and increase data throughput.

References

[1] Plummer, William W.. *TCP Checksum Function Design.* IEN-45, June 1978. ACM Computer Communications Review, April 1989.

[2] Borman, David A. *Implementing TCP/IP on a Cray Computer.* ACM Computer Communications Review, April 1989.

[3] Nicholson, A., J. Golio, D. Borman, J. Young, and W. Roiger. *High Speed Networking at Cray Research.* ACM Computer Communications Review, January, 1991.

[4] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary.* Thinking Machines Corporation, Cambridge, MA., October 1991.

[5] Fox, R. *TCP Big Window and Nak Options.* RFC 1106, June 1989.

[6] Jacobson, V., and R. Braden. *TCP Extensions for Long-Delay Paths.* RFC 1072, October 1988.

[7] Jacobson, V., R. Braden, and D. Borman. *TCP Extensions for High Performance.* RFC 1323, May 1992.

[8] Don Tolmie and Bob Beach, Editors. *High-Performance Parallel Interface - Memory Interface (HIPPI-MI).* ANSI, X3T9.3, Rev 2.6, March 1992.

[9] Don Tolmie and Ken Hardwick, Editors. *High-Performance Parallel Interface - Physical Switch Control (HIPPI-SC).* ANSI, X3T9.3, Rev 2.6, July 1992.

[10] Don Tolmie, Editor. *High-Performance Parallel Interface - Mechanical, Electrical, and Signalling Protocol Specification (HIPPI-PH).* ANSI, X3T9.3, Rev. 8.1, June 1991.

[11] Don Tolmie, Editor. *High-Performance Parallel Interface - Encapsulation of ISO 8802-2 (IEEE Std 802.2) Logical Link Control Protocol Data Units (HIPPI-LE).* ANSI, X3T9.3, Rev. 3.4, June 1992.

[12] Don Tolmie, Editor. *High-Performance Parallel Interface - Framing Protocol (HIPPI-FP).* ANSI, X3T9.3/89-013, Revision 4.3, February 1992.

[13] Hardwick, K., J. Halpern, and W. Franta. *A Link State Routing Algorithm for the Configuration of Interconnected HIPPI Switches.* Network Systems Corporation, Pre-Publication Draft, April 1991.

[14] Gigabit Networking Group. *Critical Issues in High Bandwidth Networking.* RFC 1077, B. Leiner editor, November 1988.

[15] Mogul, J., and S. Deering. *Path MTU Discovery.* RFC 1191, November 1990.

[16] O'Mally, S., and L. Peterson. *TCP Extensions Considered Harmful.* RFC 1263, October 1991.

[17] Zweig, J., and C. Partridge. *TCP Alternative Checksum Options.* RFC 1146, March 1990.

[18] McKenzie, A. *A Problem with the TCP Big Window Option.* RFC 1110, August 1989.

[19] Braden, R., D. Borman, and C. Partridge. *Computing the Internet Checksum.* ACM Computer Communications Review, April 1989.

[20] Jacobson, V. *4BSD TCP Header Prediction.* Computer Communication Review, Vol. 20, No. 2, April 1988.

[21] Minnesota Supercomputer Center, Inc. *CM-5 Technical Overview.* AHPCRC, MSCI. March 1992.

[22] Thinking Machines Corporation. *Programming the NI.* Thinking Machines Corporation, Rev. 7.1, March 1992.

[23] Leiserson, C., et. al. *The Network Architecture of the Connection Machine CM-5.* TMC, April 1992.

[24] Clark, D., M. Lambert, and L. Zhang. *NETBLT: A High Throughput Transport Protocol.* Frontiers in Computer Communications Technology, Proc. of the ACM-SIGCOMM '87, August 1987.

[25] Jain, N., M. Schwartz, and T. Bashkow. *Transport Protocol Processing At GBPS Rates.* Computer Communication Review, Vol. 20, No. 4, September 1990.

[26] Partridge, C. *How Slow is One Gigabit Per Second.* Computer Communication Review, Vol. 30, No. 1, January 1990.

[27] Clark, D., J. Romkey, and H. Salwen. *An Analysis of TCP Processing Overhead.* IEEE Communications, Vol. 27, No. 6, June 1989.

[28] R. Braden, Editor. *Requirements for Internet Hosts - Communication Layers.* RFC 1122, October 1989.

[29] R. Braden, Editor. *Requirements for Internet Hosts - Application and Support.* RFC 1123, October 1989.

[30] Jain, R. *Divergence of Timeout Algorithms for Packet Retransmissions.* Proc. Fifth Phoenix Conference on Computers and Communications, Scottsdale, Arizona, March 1986.

[31] Karn, P. and C. Partridge. *Estimating Round-Trip Times in Reliable Transport Protocols.* Proc. SIGCOMM '87, Stowe, VT., August 1987.

[32] Zhang, L. *Why TCP Timers Don't Work Well.* Proc. SIGCOMM '86, Stowe, VT., August 1986.

[33] Renwick, J. and A. Nicholson. *IP and ARP on HIPPI.* RFC 1374, October 1992.

[34] Army High-Performance Computing Research Center. *The Hitchhiker's Guide to the CM-5.* Draft Copy, AHPCRC, March 1992.

[35] Reynolds, J. K., and J. Postel. *Assigned Numbers.* RFC 1060, March 1990.

[36] Caceres, R. *Measurements of Wide-Area Internet Traffic.* University of California at Berkekey Technical Report UCB/CSD 89/550. December, 1989.

[37] Caceres, R., P. B. Danzig, S. Jamin and D. J. Mitzel. *Characteristics of Wide-Area TCP/IP Conversations.* Proc. ACM SIGCOMM '91. September, 1991.

Appendix D

Thomas, Joseph P., Cavanaugh, John D. and Salo, Timothy J., *Design of a Very High-Speed Remote File System,* Minnesota Supercomputer Center, Inc., 1992.

Design of a Very High-Speed Remote File System

Joseph P. Thomas
John David Cavanaugh
Timothy J. Salo
Minnesota Supercomputer Center, Inc.*

December 14, 1992

Abstract

A new application for gigabit networks, the network supercomputer, is described. Requirements for a very high-speed remote file system to support the network supercomputer are given, and a high-level design is described which meets those requirements.

1 Introduction

The advent of gigabit-per-second wide-area networks (which we refer to as *gigabit networks*) will enable a variety of applications that have previously been infeasible. While it will be some time before all the implications of this technology become apparent, some applications can be envisioned.

One such application is the *network supercomputer*. We use this term to describe a high-performance computing system which is comprised of geographically distributed components communicating with each other via a gigabit network.

The components of high-performance computing systems have, until now, been collocated in the interest of overall system performance. The speed of the channels connecting components of a supercomputer system (for example, the channels connecting mass storage to the processor) has a great effect on the overall performance of the system. Collocation was mandated by the order-of-magnitude differences in speed between computer channels and wide-area networks (e.g., compare an 800 Mb/s HIPPI channel to a 1.5 Mb/s T1 telephone line, or even a 45 Mb/s T3 line). Geographic distribution has been impractical because, prior to gigabit networks, high-performance channels were generally limited to a distance of a few tens of meters.

Gigabit networks, which operate at speeds comparable to those of supercomputer channels, promise to alter the way that networks are used with high-performance computing systems; gigabit networks will enable network supercomputers.

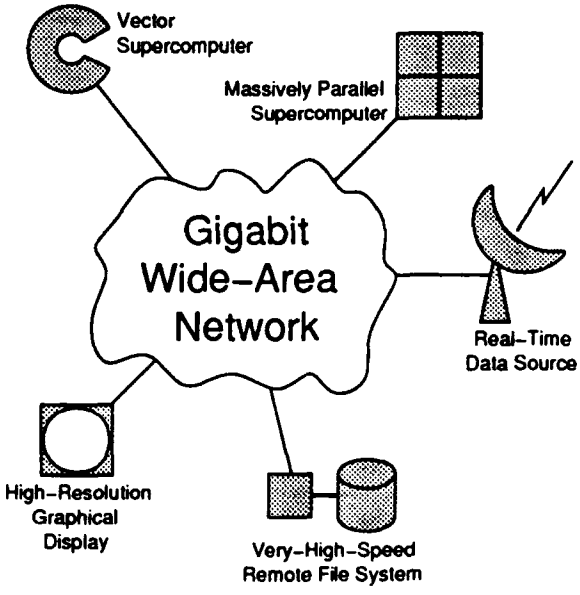

Figure 1: Network Supercomputer

A network supercomputer might be composed of

*The research described herein was sponsored by DARPA through the U. S. Army Research Office, Department of the Army, contract number DAAL03-91-C-0049. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by DARPA or the U. S. Army.

a number of high-performance components communicating via a gigabit network, as shown in figure 1. Some or all of the following components might be configured into a network supercomputer:

- A vector supercomputer
- A massively parallel supercomputer
- A shared, very high-speed remote file system
- A high-resolution graphical display
- A source of real-time data, perhaps at very high speed

The network supercomputer will enable new methods of high-performance computing. A few of the new computational methods which might be enabled include:

- **A rapidly configured high-performance system:** A researcher could "construct" a supercomputer tailored to his or her specific computational needs by interconnecting network-attached high-performance components. For example, a researcher might dynamically "construct" a network supercomputer by using a gigabit network to connect a vector supercomputer at a university campus with a prototype massively parallel supercomputer at a vendor's laboratory. These processors might process data (e.g., satellite imagery data only recently received by the ground station) from a shared, very high-speed online archive located at a third site. The researcher could be working at a fourth location which has a high-resolution graphical display.

  The network supercomputer would "exist" only for the duration of the computations. When the researcher's work was complete, the network connections which "created" the network supercomputer would be dropped.

- **Shared, very large, and possibly time-critical data archives:** Some databases are currently difficult (or impossible) to share either because of their size or the time-critical nature of their data. Making copies of these databases is not practical: the size of some very large data bases makes copying impractical, and the data in time-critical databases could be out of date by the time a copy was made. A shared very high-speed remote file system, however, would enable researchers to use data from remote locations as if they were local. The gigabit network would minimize the degradation of performance caused by accessing the data from a remote location.

- **Rapidly prototyped mixed-architecture supercomputers:** Gigabit networks will enable researchers to quickly prototype high-performance computing systems composed of computational elements with different architectures. For example, a mixed-architecture network supercomputer could be constructed by connecting a vector processor at one location with a massively parallel processor at another. More important, the mixed-architecture network supercomputer might include unique machines which could not be easily collocated with other supercomputers (e.g., one-of-a-kind or prototype machines).

- **Integration of multiple archives:** A network supercomputer could be "constructed" with connections to multiple large remote databases. The supercomputer could integrate data from these archives, perhaps displaying the result in graphical format. For example, a network supercomputer could combine air temperature data from one location with ocean surface temperature data from a second location to assist a researcher in formulating a model of the effects of air-water interaction on weather systems.

One fundamental limitation on the performance of a network supercomputer is imposed by propagation delay. Network supercomputer components could be separated by wide distances, so there would be a delay in communication while signals propagate between them (at the speed of light, a signal takes approximately 15 ms to cross the USA from coast to coast). Some activities would be affected more than others. For example, an algorithm that required a lot of short, synchronous communication between machines would be affected more than one which sent large messages without requiring synchronization.

Most of the components of the network supercomputer are now available, or are under construction. Both vector and massively parallel supercomputers are available, and they already have network interfaces operating at high speed (e.g., HIPPI interfaces at 800 Mb/s). Experimental gigabit networks are being constructed, and are the focus of much research.

The component that remains to be studied and constructed is the very high-speed remote file system. Requirements and design criteria for such a file system are discussed in the following sections.

2 Requirements

This section defines the requirements that a very high-speed remote file system (VHSRFS) must meet in order to be useful in a gigabit network (e.g., as part of a network supercomputer). A very high-speed remote file system must have these attributes:

- Very high speed
- Shared
- Expandable
- Standard interfaces
- Cost-effective

Very high speed: There are two speeds which are important to the VHSRFS. The first is the I/O speed of supercomputers. Supercomputers typically have very high-speed I/O channels, for example, HIPPI channels at 800 or 1600 Mb/s. Since supercomputers are envisioned to be important users of the VHSRFS, it should be fast enough to allow supercomputers to access data at supercomputer speed.

The second is the effective speed of wide-area networks. Although the fastest data links in common use today are 45 Mb/s, faster links are on the horizon. It will not be long before data networks at 155 Mb/s and even 622 Mb/s are commercially available. The VHSRFS should be fast enough to be able to fully utilize the capacity of these new high-speed networks.

Therefore, the VHSRFS must be able to transfer data to supercomputers across the network at 600–800 Mb/s.

Although a gigabit network can match the speed of a supercomputer I/O channel, it will have a longer propagation delay, since the length of the channel is measured in hundreds of kilometers instead of in meters. The VHSRFS must implement measures designed to minimize the effect of the longer propagation delay.

Shared: The VHSRFS must be shared simultaneously among a number of network-connected machines. These machines may be of different architectures, and may use different operating systems. Further, they may want to make use of the VHSRFS at different times or all at once.

There are two models for remote mass storage:

- A remote disk drive
- A remote file system

The remote disk drive relies on the user's system to provide meaning and structure to the data. User requests to a remote disk drive are in the form of "read or write data blocks M thru N". The extent of processing handled by the remote system is limited to the actual data transfer. While this approach simplifies the remote disk's design, it introduces a serious drawback: every host which uses the remote disk must have the same idea of its content.

In the case of a VHSRFS server, hosts make requests for file blocks as opposed to disk blocks. Since only one machine, the server, needs to know the disk layout, information about data allocation and layout can be hidden from the client machines by the server. In fact, the layout can change without the clients ever being aware. Another advantage of a file system model versus a disk drive model is that the file system can handle issues of data consistency. Where necessary, the file server can arbitrate access and can prevent destruction of data by one host while another is still accessing it.

Expandable: The VHSRFS must be available in a range of sizes. Some databases might need only

a few gigabytes of storage, while others might need a terabyte or more. Adding more storage (e.g., when a database overflows the available storage) to a VHSRFS should be straightforward.

The largest VHSRFS must be able to accommodate databases of tens to hundreds of terabytes.

Standard interfaces: One attraction of the network supercomputer is its ability to take many forms by using many different machines; it is crucial that all these different machines be able to make effective use of the VHSRFS.

To allow this, the VHSRFS must present a standard interface to the network. It must use standard file system protocols and standard network protocols (e.g., ATM, AAL5, IP, TCP, UDP). It must attach to the network through a standard physical interface (e.g., HIPPI, SONET). This will allow the VHSRFS to converse with supercomputers using standard software and readily-available network technologies.

Cost-effective: The VHSRFS must provide a cost-effective method of storing data. It must not be too expensive (e.g., compared to existing supercomputer disks), while still providing performance, expandability, and capacity as noted above. The VHSRFS can help make the overall networking environment more cost-effective by reducing the need for redundant copies of data (and the need for storage for those copies).

3 Design

This section proposes a high-level VHSRFS design capable of meeting the stated requirements. We propose that the VHSRFS be built using a large-memory, high-performance processor with HIPPI-connected RAID (Redundant Arrays of Inexpensive Disks) mass storage technology. This design utilizes readily-available hardware and reduces the risks during system integration.

3.1 Architecture

The proposed VHSRFS architecture (Figure 2) consists of:

- A high-performance processor
- Large memory
- RAID storage subsystem

Most of these parts are available in commercial products. High-performance processors (e.g., superminicomputers, or small supercomputers) are offered by a number of vendors. They are available with enough memory to meet the requirements of a VHSRFS. RAID storage subsystems are offered commercially (e.g., by Maximum Strategy), and HIPPI interfaces for them are available.

One part which is missing is a network interface which can attach the VHSRFS to a high-speed wide-area network. HIPPI interfaces are available on high-performance processors, but there is currently no way to attach such a processor to a wide-area network. A gateway with both HIPPI and SONET/ATM interfaces is not currently available (although efforts in this arena are underway), and SONET/ATM interfaces at 622 Mb/s are not available for high-performance processors.

3.1.1 High-Performance Processor

We claim that the VHSRFS server must have processing power to meet the requirements of shared access, standard interfaces, and very high speed.

The requirement to support shared access is met by implementing a file system protocol designed around the IEEE Mass Storage System Reference Model (MSSRM) [1]. The issues of selecting and implementing file systems protocols are discussed in a companion paper, *Technologies for Gigabit Distributed File Systems* [4]. The MSSRM enumerates several tasks requiring processing capability on the part of the server. These include:

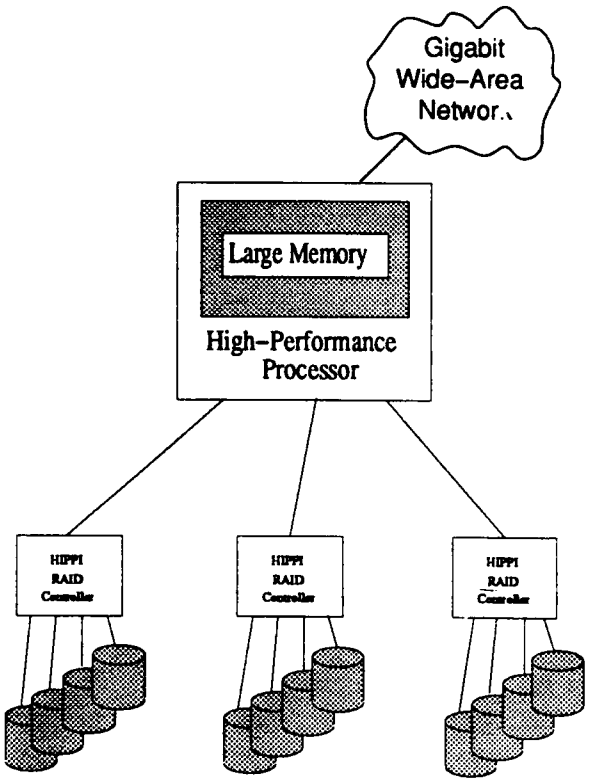

Figure 2: VHSRFS Server

- **Mapping client names to VHSRFS storage:** The MSSRM provides a global name space which is location independent. The *Name Server* module in the MSSRM maps names to physical storage locations. The name mapping allows the VHSRFS server to rearrange storage for space or access time needs independent of client knowledge.

- **Data buffering:** The VHSRFS server can improve performance by implementing buffering. When clients request data, the server can attempt to predict the clients next request and "read ahead" some amount of data. Output data can also be buffered in an attempt to write one large buffer in favor of several small buffers.

- **Access arbitration:** When several clients attempt to access the same piece of data, the VHSRFS server must implement an access policy to guarantee data consistency. This is important for cases where one client is writing data that a second client is attempting to read.

- **Access security:** In a shared environment, the MSSRM needs to provide data security such that unauthorized clients cannot gain access to sensitive data.

The requirement for standard interfaces states that the VHSRFS will present a standard interface to the network, i.e. it will use the TCP/IP protocols. It is recognized that networks with large *bandwidth* $\times$ *delay* products, called Long Fat Networks (LFNs, pronounced "elephan(t)s"), introduce TCP performance problems (see [2]). As the network community researches these problems, protocol changes will be introduced to provide performance enhancements. The use of a processor within the VHSRFS will allow the system to rapidly adapt and provide higher overall system performance.

The need for a high-performance processor is a result of the network supercomputer model. We observe that one likely class of users of a large VHSRFS system will be supercomputer-class machines. These machines will be capable of sustaining very large, high-speed data transfers. If the VHSRFS system is to maintain pace with these machines, especially when multiple clients are making simultaneous requests, the VHSRFS server must be capable of accepting and transmitting data to clients at supercomputer channel speeds (600-800 Mb/s). Furthermore, while client requests are received and replied to over the network, the server must be accessing the storage repository at these same speeds. This model implies that the VHSRFS server must be able to interface to a high-speed network based on interfaces such as HIPPI or SONET/ATM, and must also support very large amounts of data storage at supercomputer transfer rates (such as HIPPI connected disks). We make the observation that a general purpose supercomputer, that is, a supercomputer with fast scalar performance but without the need for vector processors or parallelization, meets the above requirements.

3.1.2 Large Memory

The VHSRFS server requires a very large memory pool in order to perform at the required

level. Large amounts of memory are required for buffering, both for the storage subsystem and the network interface.

The server's performance will be limited by the time it takes to read and write data from the storage system. Common file system techniques employ buffering to reduce this time. When read requests are received, the server can often correctly anticipate that the next read request from this client will be the data following the data just read. If the server were to read the data ahead, that is, read into memory more data than was requested for the transfer, there is a good chance that the next read request could be satisfied from buffer memory rather than requiring storage access. In the same manner, the server can anticipate that a write request will follow the last write request and delay physically writing the data to storage until a larger buffer has been filled.

Similarly, file system throughput can be enhanced by buffering the name server translations. As client requests are received, the name server module maps client names into storage locations. By caching frequently-used and most-recently-used translations, translations can be satisfied from table lookups rather than by a (perhaps) long computational process.

The network interface also requires large amounts of buffer storage. The network supercomputer model has multiple supercomputers accessing a small number of shared data archives. This implies that the VHSRFS must first support a large number of client connections, and that data transfers will tend to be very large to satisfy supercomputer needs. From a protocol standpoint, this leads to the need for a large amount of memory buffer to support receive and transmit queues, fragmentation and reassembly queues, and retransmission queues. If we consider some of the proposed TCP extensions to improve performance on LFNs [2], then we also have additional needs for large memory. One of the observed limitations is the inability to keep the network path fully utilized. The solution, that of increasing the window size, means that an even larger amount of data may be queued per each network connection. Congestion and data loss in the network also leads to the necessity for large memory. If any one cell (ATM layer) or fragment/packet (IP layer) is lost, the entire window of data may need retransmission. If the window has been scaled to improve throughput, the data required to be held in memory similarly increases.

How much memory is required for all these buffers? We estimate that file system buffers are likely to absorb about a megabyte of memory per active user. We estimate that the retransmission queue for the network interface will require 3–5 megabytes, and that the transmit and receive queues could require another megabyte per active user. For a moderate number of users (e.g., 10 users), the VHSRFS would require around 25 megabytes of buffer storage.

3.1.3 RAID Storage Subsystem

A RAID-based storage subsystem is recommended because of its ability to provide high-performance, fault-tolerant data storage at a reasonable cost.

RAID is a concept which an array of disk drives working in parallel is used to improve data throughput, improve reliability, and reduce cost. Whereas a single disk might provide an 8-bit data path with a 2 MB/s data throughput, four of those same drives working in parallel can provide a 32-bit data path with 8 MB/s of data throughput. Since a larger number of components leads to a higher risk that one of those components will fail sooner then a single component would, RAID also employs some type of data redundancy to provide system level fault tolerance. This redundancy means that a RAID subsystem can maintain a high MTBF, even though the MTBF of the sum of its parts is low.

There are five different types, or levels, of RAID.

RAID level 1 or RAID 1, provides fault tolerance through disk mirroring. At this level, two identical copies of the data are kept on two physically separate disks. Failure of one disk leaves the "mirrored" copy intact. The disadvantage of RAID 1 is 50 percent utilization. For every disk used to store data, another full disk is needed to mirror it. This means that a level 1 RAID subsystem is expensive, and does not provide a significant improvement in throughput.

RAID 2 uses data striping with an interleaved Hamming code. Here, each bit of data is written to a separate disk. Interleaved with the data bits is a Hamming code capable of correcting single bit errors and detecting double bit errors. Disadvantages of RAID 2 include the need to compute complex Hamming codes, and the large number of check bits needed. With 10 data drives, four check bit drives would be needed, for 71 percent utilization. As the number of data bits drops below 10, the utilization can drop below the 50 percent level.

RAID 3 draws on the observation that many of todays drives already incorporate ECCs (error-correction code's) making Hamming codes redundant. Instead, RAID 3 stores only enough information to correct a single failed drive. RAID 3 uses bit level striping with one check drive containing an XORed parity bit. If any one drive fails, the data can be reconstructed by XORing the remaining data bits with the check bit. One disadvantage of RAID 3 is that each drive transfers one sector for a N-wide data transfer unit. As N starts to grow to even small numbers, the transfer unit size grows and small I/O begins to lose performance. The penalty is greater for small writes because the entire transfer unit must be read, a small portion modified, and the entire transfer unit written back to disk.

RAID 4 attempts to overcome the read performance limitation of RAID 3 by using sector striping instead of bit striping. With sector striping, unrelated sectors may be read off different drives simultaneously. That is, when reading, the drives in the array may be at different sector locations simultaneously. RAID 4 still imposes the write penalty since there is only one check disk. This limits the system to performing write operations one at a time.

RAID 5 removes the write bottleneck from RAID 4 by distributing the check disk function among several drives. In RAID 5, disk one may be the check disk to sector one while disks two through five are the data disks. The check disk for sector two might be disk three, while disks one, two, four and five are the data disks. Since the XOR nature of the check disk requires that only two disks be written too, the data disk and the check disk, if the write operation involves different disks, several write operations may take place simultaneously.

We recommend that a VHSRFS employ a level 5 RAID storage subsystem. It offers the best performance, and commercial products are available (e.g., from Maximum Strategy).

3.2 Satisfying the Requirements

This section briefly covers the stated requirements and how they are met by the proposed VHSRFS design.

Very high-speed: A VHSRFS design requires high speed in multiple areas. Tasks to be simultaneously carried out at supercomputer speeds include:

- Accept input requests and data from the network
- Perform file system protocol processing
- Transfer data to and from mass storage
- Send responses and data to the network

As wide-area networks capable of supporting supercomputers (i.e. 600–800 Mb/s) are implemented, the overall demands on the server also increase.

The selection of a high-performance processor with large amounts of memory allow the VHSRFS server to concurrently handle high network demands while supporting very fast storage access.

Shared access: To support the shared access requirement, a high-performance processor is recommended. Use of a processor allows the VHSRFS to become a programable system. This programability is the basis for implementation of network and file system protocols, something that is not possible on high-speed disk controllers. Programability allows for the growth and development of protocols as the VHSRFS concept matures. It is recognized that current TCP/IP protocols have limitations for high *bandwidth* × *delay* networks. Extensions have been proposed but will need to be implemented to surmount these issues. Similarly, there are likely to be issues in file system performance which will need to be dealt with.

Standard interfaces: A goal of the VHSRFS is to provide a new service to existing users. The use of standard protocols, both network and file system, will achieve this goal. The VHSRFS needs to support the TCP/IP protocol suite for network access. Physical interfaces need to support the high bandwidth of the network and may include both HIPPI and SONET/ATM interfaces. A standard file system protocol will allow the VHSRFS to be shared among a large number of remote hosts. Standard interfaces also allow connection with more traditional networks, such as FDDI, in order to provide access to tape drives and other low-performance storage media.

Expandable: A goal of the VHSRFS is to support databases of tens to hundreds of terabytes in size. RAID technology allows the storage capacity to grow by the addition of inexpensive disks. Use of RAID technology also allows the storage media to provide the high-bandwidth data transfers required. As the demands on the VHSRFS server grow, the use of a large-memory, high-performance processor will allow I/O bandwidth to grow, along with providing support for higher demands on the file system and network protocols.

Cost-effective: Cost objectives can be met with the selection of RAID storage supported by a high-performance processor using standard interface technology. The use of RAID provides a reliable, low-cost, high-bandwidth storage media. The VHSRFS server requires a processor capable of maintaining supercomputer speeds, but does not require the special hardware, such as vector processing, often associated with supercomputers. By selecting TCP/IP as the network protocol, and using HIPPI as a channel both for the network and for the storage system, costs can be controlled by not requiring the development or purchase of unique, one-of-a-kind, software or hardware.

4 Summary

Gigabit networks promise to enable a number of applications that are currently infeasible. To support these applications, a new shared very high-performance remote file system is required. A VHSRFS should be very fast, sharable, expandable, have a standard network interface, and be cost-effective. We propose a high-level design for a VHSRFS based on a large-memory, high-performance processor and RAID technology. This design has provisions to meet these requirements. Most of the components of the design are available commercially.

A Acronyms

AAL	ATM adaptation layer
AAL5	ATM adaptation layer 5
Gb/s	gigabits per second
HIPPI	high-performance parallel interface
I/O	input/output
IP	Internet Protocol
Mb/s	megabits per second
MB/s	megabytes per second
ms	milliseconds
MSSRM	mass storage system reference model
MTBF	mean time between failures
RAID	redundant array of inexpensive disks
SONET	synchronous optical network
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VHSRFS	Very High-Speed Remote File System

References

[1] Coleman, S., and S. Miller, Editors. *Mass Storage System Reference Model.* IEEE, Ver. 4, May 1990.

[2] Jacobson, V., R. Braden, and D. Borman. *TCP Extensions for High Performance.* RFC 1323, May 1992.

[3] Alford, Roger. *Disk Arrays Explained.* BYTE, October 1992.

[4] Spengler, Michael K., and Timothy J. Salo. *Technologies for Gigabit Distributed File Systems.* Minnesota Supercomputer Center, Inc.